DOCUMENT RESUME

ED 337 126 IR 015 112

AUTHOR Whitehall, Bradley Lane
TITLE Knowledge-Based Learning: Integration of Deductive and Inductive Learning for Knowledge Base Completion.
INSTITUTION Illinois Univ., Urbana. Dept. of Computer Science.
REPORT NO UILU-ENG-90-1776; UIUCDCS-R-90-1637
PUB DATE Oct 90
NOTE 163p.; Ph.D. Dissertation, University of Illinois at Urbana-Champaign.
PUB TYPE Dissertations/Theses - Doctoral Dissertations (041) r

EDRS PRICE MF01/PC07 Plus Postage.
DESCRIPTORS Algorithms; *Computer System Design; Deduction; Engineering Education; *Expert Systems; Higher Education; Induction; Knowledge Level; Research Methodology; *Systems Development
IDENTIFIERS *Machine Learning

ABSTRACT

In constructing a knowledge-based system; the knowledge engineer must convert rules of thumb provided by the domain expert and previously solved examples into a working system. Research in machine learning has produced algorithms that create rules for knowledge-based systems, but these algorithms require either many examples or a complete domain theory (often in the form of rules). In many real-world situations, only a limited number of examples and an incomplete domain theory are available. This thesis presents a new paradigm for machine learning--knowledge-based learning (KBL)--which combines the strengths of empirical learning with the strengths of explanation-based learning to overcome previous limitations. Two systems--KBL0 and KBL1--are used to illustrate the new paradigm. These systems have been designed and implemented to work with domains requiring a representation of real numbers and mathematical formulas, such as engineering. This research has shown not only that it is possible to use a domain theory to guide induction using examples, but that when there are few examples available compared to the size of the problem space, the resulting rules are more accurate and stable than those from pure empirical techniques. In addition, knowledge-based algorithms free the user from selecting relevant examples and attributes for learning by using an incomplete domain theory to determine where knowledge needs to be added. A problem unsolved by the current domain knowledge helps to determine where new knowledge needs to be incorporated into the domain theory and what the context is for the learning. The context is used to select relevant examples from an example base and to reduce the number of attributes needed during the induction. With the control structure provided by knowledge-based systems, inductive learning can be used to extend an existing knowledge base. (69 references) (Author/DB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

IR

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED337126

REPORT NO. UIUCDCS-R-90-1637 UILU-ENG-90-1776

KNOWLEDGE-BASED LEARNING: INTEGRATION OF DEDUCTIVE AND INDUCTIVE LEARNING FOR KNOWLEDGE BASE COMPLETION

by

Bradley Lane Whitehall

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

William J. Kubitz

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR015112

October 1990

BEST COPY AVAILABLE

©Copyright by
Bradley Lane Whitehall
1990

KNOWLEDGE-BASED LEARNING: INTEGRATION OF DEDUCTIVE AND INDUCTIVE LEARNING FOR KNOWLEDGE BASE COMPLETION

BY

BRADLEY LANE WHITEHALL

B.S., Northern Illinois University, 1983
M.S., University of Illinois, 1987

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1990

Urbana, Illinois

KNOWLEDGE-BASED LEARNING: INTEGRATION OF DEDUCTIVE AND INDUCTIVE LEARNING FOR KNOWLEDGE BASE COMPLETION

Bradley Lane Whitehall, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1990
Stephen C-Y. Lu, Advisor

To learn effectively, a system needs to use all the knowledge that is available. Explanation-based learning and similarity-based learning operate over a domain theory and a set of examples, respectively, but neither approach makes extensive use of both forms of knowledge. Many problems in engineering and other areas can provide a learning system with an incomplete domain theory and a limited set of examples. Knowledge-based learning uses knowledge in both forms to learn knowledge missing from the domain theory.

The knowledge-based learning approach is illustrated with two systems, KBLØ and KBL1. These system have been designed and implemented to work with domains requiring a representation of real numbers and mathematical formulas, such as engineering.

This research has shown, not only that it is possible to use a domain theory to guide induction using examples, but that when there are few examples available compared to the size of the problem space, the resulting rules are more accurate and stable than those from pure empirical techniques. In addition, knowledge-based learning algorithms free the user from selecting relevant examples and attributes for learning by using an incomplete domain theory to determine where knowledge needs to be added. A problem unsolved by the current domain knowledge helps to determine where new knowledge needs to be incorporated into the domain theory and what the context is for the learning. The context is used to select relevant examples from an example base and to reduce the number of attributes used during induction. With the control structure provided by knowledge-based systems, inductive learning can be used to extend an existing knowledge base.

To my parents — Floyd A. and Bonnie L. Whitehall — for all their love and support.

Acknowledgements

I would like to thank Robert Stepp for his insight, wisdom, and guidance on this research. He has taught me how to do research and how to recognize progress.

I would also like to thank Stephen Lu for guiding this research, and for helping me to realize its potential. His enthusiasm and energy are contagious. He is always driving to apply research to real problems and to discover concepts that will impact the way the world works. He could see the horizon, when I was unable to see over the next hill.

I would like to thank Gerald DeJong for being on my committee and at the last minute saving me from administrative red tape. He has influenced this work in many ways, especially by allowing me to participate in his group's weekly meetings at CSL.

Professors David Wilkins, Seth Hutchinson, and Sylvian Ray served on my final examination committee, providing suggestions for improvements and insights on learning. Professor Gregg Collins served on my prelim committee and provided valuable comments on the work.

Over the years, there have been conversations with many people on different topics. Ray Mooney, my first officemate, has clarified more issues for me than I care to recall. Jude Shavlik and Alberto Segre were always in the office and willing to explore new issues and ideas. Other members of the Explanation-Based Learning group made those weekly meetings interesting, Steve Chien, Scott Bennett, Shankar Rajamoney, and Melinda Gervasio.

The members of Professor Stepp's research group, Larry Holder, Bob Reinke, Bharat Rao, Diane Cook, and Bruce Katz have been very helpful during my graduate studies. Always willing to listen to an unpolished talk and discuss research, these people have had a large impact on this work.

I would like to thank Carl Kadie for being the best proofreader I have ever met. But more importantly, for always being willing to discuss ideas and present different perspectives on

machine learning. The first person I met at graduate school was Carl. I expect the friendship to last many more years with lots of good ideas exchanged along the way.

I would like to thank Ken Smith for sparking my interest in database research. I was disinterested in databases when I came back to school, but through discussions with Ken it has been clear that ideas learned in the past can be used in new and exciting ways.

Prof. Lu's research group has been very helpful in providing real problems and interesting perspectives on this work.

All the people mentioned above have influenced the technical aspect of this work. Those people have also made graduate school an enjoyable experience. There are a number of people that deserve recognition even though they did not contribute to the technical aspects of this research. They added special qualities to my life that made the hard work not seem so hard.

My parents deserve more praise than I can give them. They have always been there for me. Just knowing that they support me in every decision I make has helped many times.

I would like to thank my sister, Elizabeth, for sharing her time and experiences with me while she was an undergraduate at Illinois. She always had time for a phone call and an occasional lunch. Hearing her experiences reminded me that being a graduate student is not the worst fate in the world, being an undergraduate is.

I would like to thank John Serituk for calling when I expected it the least and needed it the most. John always made me laugh regardless of the current "crisis." Thanks for all the tennis games and late night discussions.

I would like to thank the dessert crew, Nanci Vaeth, Sheldon Nicholl, Tedd Pierce, Rob Bryan, Carl Kadie, and especially the dessert maker, Elaine Chapin. I probably could have lived without the sugar, but not without the companionship and interesting conversations.

I would like to thank my racketball partners from the last couple years, Dave Luginbuhl and Gregg Gunsch. They allowed me to take out my aggressions on a little blue ball instead of the people around me.

Finally, I would like to thank the person who means so much to me, Denise Krasnowski. She has filled my stomach with delicious cookies and cheesecakes. filled my soul with hope for the future, and filled my heart with love.

Funding for this research was partially supported by a research contract from Digital Equipment Corporation, by the National Science Foundation, and by a gift from Texas Instruments, Inc. A forgivable loan from the General Electric Corporation is also gratefully acknowledged.

Table of Contents

Chapter		Page
1	**Introduction**	1
	1.1 Motivation	2
	1.2 Thesis	4
	1.3 Overview of KBL	5
	1.4 Concept Illustration	7
	1.5 Outline	8
2	**The Predecessors of KBL**	10
	2.1 Explanation-Based Learning	10
	2.1.1 EBL Structure Manipulation	12
	2.1.2 EBL Problems	13
	2.2 Similarity-Based Learning	14
	2.2.1 SBL Structure Manipulation	16
	2.2.2 SBL Problems	17
	2.3 Related Integrated Learning Approaches	18
	2.3.1 EBL with Theory Revision	18
	2.3.2 UNIMEM	19
	2.3.3 CLUSTER/CA	19
	2.3.4 OCCAM	19
	2.3.5 GEMINI	20
	2.3.6 IOE	20
	2.3.7 IOU	21
	2.3.8 TEIRESIAS	21
	2.3.9 DISCIPLE	22
	2.3.10 ODYSSEUS	22
	2.3.11 Learning from Plausible Explanations	23
	2.3.12 META-DENDRAL	23
	2.3.13 LEX	24
	2.3.14 Relation of Other Work to KBL	24
3	**Knowledge-Based Learning Paradigm**	25
	3.1 KA, ML, and KBL	27
	3.2 KBL Overview	29

3.2.1 Partial-Proof-Tree Constructor . . . 31
3.2.2 Induction Module . . . 32
3.2.3 Context Directed Rule Specialization . . . 36
3.2.4 Deductive Generalization . . . 38
3.2.5 KBL Overall Structure . . . 38
3.3 Methods for Evaluating KBL . . . 39
3.3.1 Evaluating Effectiveness . . . 41
3.3.2 Evaluating Correctness . . . 42
3.4 Summary . . . 43

4 KBLØ . . . 45
4.1 The KBLØ System . . . 46
4.1.1 Representation . . . 47
4.1.2 System Components . . . 49
4.1.3 KBLØ Summary . . . 54
4.2 Knowledge-Based Equation Generator . . . 54
4.2.1 Motivation . . . 56
4.2.2 KBEG . . . 59
4.2.3 KBEG Demonstration . . . 63
4.2.4 Summary of KBEG . . . 65
4.3 Comparing ID3 and KBLØ . . . 66
4.3.1 Testing Method . . . 67
4.3.2 Learning Results . . . 68
4.3.3 Results on Stability . . . 69
4.4 Conclusion . . . 70

5 KBL1 . . . 72
5.1 Rule-Structured Induction . . . 73
5.1.1 Problem Definition . . . 74
5.1.2 Assigning Heuristic Goodness Values . . . 77
5.1.3 Search For Generalization . . . 79
5.2 Regression of Variable Values . . . 81
5.2.1 Extensions . . . 85
5.3 Inducing Rule Consequents . . . 88
5.4 Inducing Rule Antecedents . . . 89
5.4.1 Related Work . . . 90
5.4.2 KBL1's MSG . . . 92
5.5 Specializing Created Rules . . . 96
5.6 Converting AND/OR Trees to OR Trees . . . 96
5.7 KBL1 . . . 98
5.8 Demonstration of KBL1 . . . 100
5.8.1 Single Rule Constructed . . . 101
5.8.2 Two Rules Constructed . . . 104
5.8.3 Extending an Existing Rule . . . 106
5.9 KBL1 Summary . . . 107

6 Empirical Results 109
6.1 Justifying KBL 109
6.1.1 Experimental Results 110
6.1.2 No Context Results 116
6.2 Phone Domain 117

7 Conclusion 124
7.1 Future Work 124
7.2 Final Summary 126

Appendix 129

A Engineering Test With ID3 129

Bibliography 140

Vita 146

List of Figures

Figure		Page
1.1	Range of Knowledge	2
1.2	Evolving Knowledge Base	3
1.3	Machining Domain	7
2.1	EBL Rule making Process	13
2.2	SBL Structuring of the Knowledge Base	15
2.3	SBL Rule making Process	17
3.1	Knowledge Acquisition with a Knowledge Engineer	27
3.2	Empirical Machine-Learning Approach to Knowledge-Base Construction	28
3.3	Knowledge-Based Learning Approach to Knowledge-Base Construction	29
3.4	Details of KBL Approach to Knowledge-Base Construction	30
3.5	Knowledge-Based Learning Induction Module	33
3.6	Example Partial-Proof Tree	33
3.7	KBL and Performance Element Interactions	38
4.1	KBLØ System Overview	46
4.2	Simple Partial-Proof Tree	51
4.3	Reducing Example Space	52
4.4	Knowledge Flow of KBLØ	55
4.5	KBEG algorithm kernel	61
4.6	KBLØ Binary Test Domain	66
4.7	System given 50 initial cases	68

4.8 System given 25 initial cases 69
4.9 System given 35 initial cases 69
4.10 System given 100 initial cases 70
4.11 System given 200 initial cases 70

5.1 Rule-Structured Induction 76
5.2 Factors Computation 80
5.3 Sample User Preference Values for Generalization 80
5.4 RSI Search 81
5.5 Rules from Knowledge Base 82
5.6 Partial-Proof Tree with Highlighted Border 84
5.7 Computing Possible Variable Values 86
5.8 Back-Extend 87
5.9 Determining Constant Value in KBEG-R 89
5.10 MSG Search 92
5.11 MSG Description Generation 94
5.12 Examples for MSG 95
5.13 State-Space Example 97
5.14 Search with PPT Nodes 98
5.15 Toy CAPP Knowledge Base 100

6.1 Dependent Domain Theory 111
6.2 Independent Domain Theory 112
6.3 Example Base with 25 Events 113
6.4 Example Base with 50 Events 113
6.5 Example Base with 75 Events 114
6.6 Example Base with 25 Events 114
6.7 Example Base with 50 Events 115
6.8 Example Base with 75 Events 115
6.9 No Context Difference Results with 75 examples 117
6.10 No Context Improvement Results with 75 Examples 118
6.11 Accuracy for KBL1 with `susp-from-loc` Removed from Phone Domain 122

6.12 Accuracy for KBL1 with **susp-tod** Removed from Phone Domain 122
6.13 Accuracy for KBL1 with **susp-freq** Removed from Phone Domain 123
6.14 Accuracy for KBL1 with **susp-length** Removed from Phone Domain 123

7.1 Knowledge Engineer's Effort Required to Achieve Desired Performance 126

A.1 Typical PHAST Example 130

List of Tables

Table		Page
4.1	Unclusterable Data	58
4.2	Data set used by ABACUS and KBEG	59
4.3	Simple ABACUS Example	60
4.4	Data explained by 2 formulas	64
4.5	Percent Accuracy at Episode 25	71
4.6	Percent Accuracy at Episode 10	71
5.1	KBEG-R Example	90

Chapter 1

Introduction

Constructing knowledge-based systems is difficult. The knowledge engineer must convert rules of thumb provided by the domain expert and previously solved examples into a working system. As with many projects, the 20/80 rule appears to be true in knowledge-base construction. That is, 80% of a system's functionality can be obtained in the first 20% of the knowledge engineer's effort. Obtaining the rest of the knowledge and successfully integrating it into the existing knowledge base is difficult. Tools are needed to assist the knowledge engineer in completing this task.

Research in machine learning has produced algorithms that create rules for knowledge-based systems. As input, these algorithms require either many examples or a complete domain theory (often in the form of rules). In many real-world situations, only a limited number of examples and an incomplete domain theory are available. In such cases, a learning algorithm that can use knowledge from both examples and rules is required. For a learning system to be useful, it must produce rules that can be incorporated into the existing knowledge. Previous approaches to machine learning have not been applicable to the above situation. Those approaches required either a large set of examples or a complete knowledge base. In this thesis, a new paradigm for machine learning is presented: *knowledge-based learning* (KBL) combines the strengths of empirical learning with the strengths of explanation-based learning (EBL) to overcome previous limitations. One of the keys to knowledge-based learning is its use of an incomplete domain theory to guide the selection of examples for the induction of rules. The knowledge provided

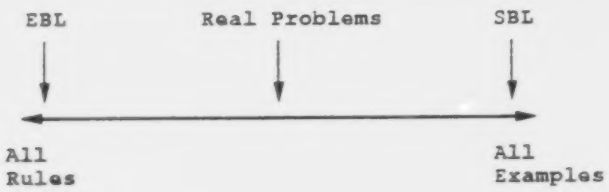


Figure 1.1: Range of Knowledge

to a KBL system functions in one of two roles; as examples, it guides an inductive process and as rules, it guides a deductive process.

1.1 Motivation

This work is motivated by the need to have learning systems that can work in realistic situations. More specifically, the work is motivated by the need to have learning systems that can deal with the complexities of engineering. The two current approaches to symbolic machine learning work under extreme conditions (Figure 1.1). Explanation-based learning requires knowledge in a form usable by the performance element, (i.e., rules). Empirical learning requires knowledge in example form. Many real problems, however, do not fit into either of these extreme situations. In engineering problems, typically both forms of knowledge are available and are used by good engineers. Engineers combine years of formal training (obtaining rules) with extensive problem-solving experience (obtaining examples). This is what makes experienced engineers valuable to an organization.

Knowledge-based learning is motivated by engineering problems on three levels:

1. Knowledge acquisition and knowledge utilization merger.

   This top-level goal differs from other learning techniques by using knowledge in multiple forms. KBL constructs rules within the context of an existing (partial) knowledge base using knowledge existing in example form. The current problem assists the system in two ways. First, it guides the selection of relevant examples to use in induction. Second, it helps the system create a rule that integrates with the existing knowledge.

2. Knowledge evolution.

   The knowledge required to solve problems changes constantly. Learning systems must

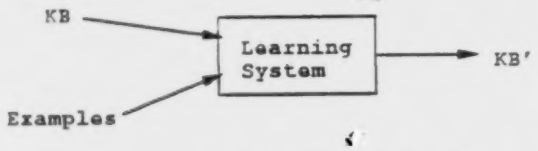


Figure 1.2: Evolving Knowledge Base

account for these changes. KBL changes the current knowledge base so that performance can improve (Figure 1.2).

3. Problem specific techniques.

   Engineering problems require a learner that can deal with real numbers (in addition to nominally valued attributes) and that can produce an infinite number of class values. KBL produces rules that incorporate discovered formulas that describe relationships between attributes.

In engineering problems it is very difficult to get all the knowledge into a form usable by the performance element (thus, the need for a learning system). On the other hand, engineering domains are complex and would require far too many examples for a purely empirical approach to be practical. Moreover, these rules do not necessarily work within an existing knowledge base.

Engineering problems also require a more complex rule representation than learning systems provide. Traditional similarity-based learning systems produce rules with antecedents that test the values of specific attributes and with consequents that assign class values (constants) to the example. In engineering domains, this is not adequate because the consequent must be allowed to assign a computed value instead of a constant. The difference is that in standard empirical learning the number of possible class values is small and known a priori. In engineering domains, the number of possible class values is infinite because a class variable can be based on a formula involving other variables. Thus, the representation needed by the learning system is more expressive. To help clarify the difference, consider a set of rules produced by traditional learning from example systems:

```
IF (dimensions ?object ?x ?y ?z) (< ?x 10) (< ?y 10) (< ?z 15) THEN
        (volume ?object small)
```

```
IF (color ?object ?color) (equal ?color red) (equal ?repaint true) THEN
        (new-color ?object red)
```

In the first rule, the size of an object is classified; in the second, the new color to paint an object is assigned. The similar rules written in KBL's representation would be:

```
IF (dimensions ?object ?x ?y ?z) (set ?foo (* ?x ?y ?z)) THEN
        (volume ?object ?foo)
IF (color ?object ?color) (equal ?repaint true) THEN
        (new-color ?object ?color)
```

The first rule illustrates how the representation allows an infinite number of classes to be defined. This rule is more memory efficient than having a specific rule for each color. The second rule assigns the value of *new-color* of an object to be the same as the current color, for any color.

1.2 Thesis

The thesis of this work is the following:

> *The capabilities of a knowledge-based system are extended by improving the knowledge base. The knowledge base can be extended and modified to provide better performance by integrating the current knowledge base and knowledge in the form of solved examples.*

A machine learning system has been constructed to test this thesis. The learning system uses the existing knowledge base to guide induction over a set of examples. The rules produced by induction are added to the knowledge base. In addition, the learning system can improve the performance element's performance by correcting over-general and over-specific rules.

There are three secondary goals:

1. *To manage the problems of completing an incomplete knowledge base with machine learning. Knowledge will be used to direct the learning of the missing knowledge. An inductive leap, however, is required. The system determines what are important attributes and which examples are relevant to solving the current subgoal.*

2. *To harvest knowledge from examples in a number of ways. Knowledge in examples will be used to guide induction of rules and identify important attributes.*

3. *To develop techniques required for the use of machine learning in engineering problems. This includes working with a representation that allows an infinite number of decision classes, manipulating real numbers, and discovering mathematical formulas.*

This thesis provides details on knowledge-based learning. The chapter describing the KBL paradigm shows why KBL requires complex search. To reduce the amount of search a number of heuristics have been developed. The induction component of this system is the major focus of this work, because it differs greatly from previous empirical learning systems. The goal of the system implementation was to produce a domain-independent learning system which is most useful when:

- An initial knowledge base exists. It may be incomplete.
- Examples from the domain exist with solutions.
- The performance element uses a representation requiring an infinite number of decision classes.

1.3 Overview of KBL

A knowledge-based learning system should be as deductive as possible when the situation warrants. For example, if the system has complete and accurate domain knowledge and unlimited time to perform the learning task, deductive learning should be used. At the other extreme, if all the knowledge is in the form of examples rather than rules, an empirical approach to learning is required. Handling the two extreme circumstances, however, does not make a learner a *knowledge-based* learner.

Knowledge-based learning systems are different from systems that have been researched previously. The main difference is that knowledge-based systems must be able to manipulate two distinct forms of knowledge:

- rules
- previously solved examples

The goal of a KBL system is to learn with any mix of these two forms of knowledge. First, the system must show deductively from a set of domain specific rules that a specified example is a member of a concept class. Thus, the system needs an inference engine similar to those used in knowledge-based systems and EBL systems. Second, the system must work with, and make inductive inferences from, presented examples. In addition these two components of the knowledge-based learner must work with each other. KBL integrates these components in a unique way.

This requirement leads to important modifications to the components. The inference engine must work with rules that might not be 100% correct and complete. The rules could have either been given to the system or been induced earlier by the inductive component. The inference engine must use these rules to construct a *partial-proof tree*. Though the system is chaining through rules, the proof constructor might not be purely deductive. The inference engine must also recognize when a critical rule is missing and when induction might be useful. When control is transferred to the inductive component, the system must be able to guide the search for relevant examples and important attributes. This can be accomplished by determining the current goals of the deductive process and communicating those goals (in an appropriate form) to the inductive process. The inference engine must also recognize that it has a limited amount of time available to learn a specific concept. It must determine when to stop searching for a particular concept, how many examples will be needed to induce a rule of the desired accuracy, whether it has time to start an inductive process, and so on. Thus, the inference engine must be capable of much more than simply backward chaining through complete, accurate rules.

Likewise, the inductive component is much more complex than previous inductive systems. First, the system needs to be able to identify relevant examples from the large data set of examples that have been presented. Previous inductive learning systems have considered most of the examples relevant; some examples can be treated as noise. Unlike previous learning-from-examples (LFE) systems, the learner must take into account the context of the current problem and use this context to help collect examples and determine which attributes are most relevant.

To summarize, knowledge-based learning has a number of capabilities beyond those of current machine learning systems:

- The inference engine is both deductive and inductive.

```
machine(?x ?y) ← (feature ?x ?f) (size ?x ?s) (machine-type ?x ?f ?s
   ?y)
(feature ?x ?f) ← (feature-of ?y ?x) (feature-name ?y hole) (= ?f
   hole)
(feature ?x ?f) ← (feature-of ?y ?x) (feature-name ?y slot) (= ?f
   slot)
(size ?x ?s) ← (width ?x ?w) (< ?w 10) (= ?s small)
(size ?x ?s) ← (width ?x ?w) (≥ ?w 10) (= ?s big)
(machine-type ?x ?f ?s ?y) ← (equal ?f slot) (equal ?s narrow) (= ?y
   end-mill)
```

Figure 1.3: Machining Domain

- The system uses two forms of knowledge (rules and examples).
- The system, not the user, must select the examples used in induction.
- The system determines which attributes are relevant to the induction problem.
- The system induces mathematical relations used to assign values for a subgoal.
- The system determines where knowledge is missing in the knowledge base.

1.4 Concept Illustration

To clarify the ideas presented in this thesis, a simple motivating example is now presented. If provided with an incomplete domain knowledge and a set of examples, a KBL system finds a rule to help complete the domain knowledge. The system uses the partial proof constructed in solving a problem to guide the induction of the new rule. Let the initial knowledge base contain the rules in Figure 1.3[1]. This simple domain returns the type of machine used to construct the feature when given a part. There are four possible machining values: `end-mill`, `drill`, `2-pass-end-mill`, `bore-drill`. If the system is given the facts:

```
(feature-of x x1) (feature-name x1 hole) (width x 7)
```

and asked to solve `(machine-type x ?y)` no solution would be found. A KBL system would

[1]The = predicate binds the variable of the first argument to the value of the second argument.

look at the example base and notice which examples are relevant to problem. So if the example base contained the example/solution pairs:

```
A: (feature-of a a1) (feature-name a1 slot) (width a 3) /
       (machine a end-mill)
B: (feature-of b b1) (feature-name b1 hole) (width b 3) /
       (machine b drill)
C: (feature-of c c1) (feature-name c1 slot) (width c 12) /
       (machine c 2-pass-end-mill)
Z: (feature-of z z1) (feature-name z1 hole) (width z 9) /
       (machine z drill)
```

The examples B and Z would be used to help induce a new rule for the problem because they meet the constraints of the rules all solved for the problem:
`(feature-of ?y ?x) (feature-name ?y hole) (= ?f hole) (width ?x ?w) (< ?w 10)`
Using those examples the system could induce the new rule:

```
(machine-type ?x ?f ?s ?y) ← (feature-of ?y ?x) (feature-name ?y hole)
   (width ?x ?w) (< ?w 10) (= ?y drill)
```

Because the partially solved problem determines which examples are useful in learning, KBL systems can require fewer examples when inducing a rule.

1.5 Outline

The rest of this thesis explains these aspects (and implementations) of knowledge-based learning and how they are similar to and different from previous learning systems. The second chapter reviews some of the known limitations of the two main symbolic learning approaches, explanation-based learning (EBL) (or analytical learning) and similarity-based learning (SBL) (or empirical learning). This is important because the goal of knowledge-based learning is to overcome the limitations of these systems and provide a new approach to machine learning. A discussion compares and contrasts EBL and SBL, facilitating an understanding the synthesis of the two processes into a single coherent process. This chapter also describes other systems combining empirical and analytical approaches. The third chapter explains the knowledge-based paradigm in general. The fourth chapter presents KBLØ, a knowledge-based learning

system that demonstrates some of the concepts presented in the first three chapters. Empirical evaluation of this system shows that the KBL method can be used to improve the performance of a learning system. The fifth chapter describes a more complex KBL system. KBL1 incorporates the *rule-structured induction* method to allow the system to make more use of existing rules in the knowledge base. Results of experiments with the KBL1 system are presented in Chapter 6. The final chapter describes future work, and provides concluding remarks.

Chapter 2

The Predecessors of KBL

Knowledge-based learning unifies the tasks performed by EBL and SBL via the operations they perform on a proof structure. Briefly, EBL and SBL create new rules for a performance system. EBL creates new rules by chunking intermediate rules that are used together in proving a concept. SBL creates new rules by making inductive leaps about what antecedents are required for a consequent.

In this chapter, the systems upon which KBL is built are explained. Knowledge-based learning is an integration of ideas based in explanation-based learning and similarity-based learning. EBL and SBL contain complementary strengths which make integrating the two systems a logical step in building better learning systems. The next section describes EBL. The presentation focuses on aspects of the paradigm important for KBL. The second section describes SBL and its aspects considered important in the integration with EBL as the basis for knowledge-based learning. Both EBL and SBL researchers have realized potential benefits of combining these approaches. In the third section, previous work done in combining EBL and SBL is presented.

2.1 Explanation-Based Learning

Conceptually, explanation-based learning [Mitchell *et al.*, 1986, DeJong and Mooney, 1986] is a two-step process that acquires rules that reduce problem solving time. EBL systems perform *deductive generalization*. These systems produce macro-operators that chunk knowledge

acquired from proving an example is a member of a specified concept class. The goals of EBL systems are:

- Improve problem-solving performance

  This is the major goal of EBL systems. They cannot produce new knowledge, that is, the new rules do not allow the system to perform previously unsolvable tasks. But the system can perform tasks more efficiently with the chunked knowledge of the new rules.

- Understand observation

  For the system to build a new rule it must thoroughly understand the given example.

- Generalize experience

  An EBL system produces rules generalized to the extent that the deductive knowledge allows. A major aspect of these systems is that they can prune away irrelevant portions of the proof tree when constructing the tree.

The first step of an EBL system is to build a deductive proof showing an example is a member of a concept class. EBL systems often use a standard theorem prover to build the proof tree. EBL's second step is to generalize the proof tree so that the constructed rule applies to examples other than the one seen. At its simplest, this generalization involves turning constants into variables, but more effective systems generalize at higher levels of the proof tree. Explanation-based generalization (as presented in [Mitchell *et al.*, 1986]) requires the following four components:

- Goal Concept

  A concept description of what is to be learned.

- Training Example

  An example of the concept to be learned. DeJong and Mooney [DeJong and Mooney, 1986] list many possible sources for this example. The example could be given by an instructor or observed by a passive learning system watching an expert. If the learning system has access to a competent problem solver, the training example could be the computed solution to a problem. If the example solution is given or observed, the system should check for inefficiencies because it might not be optimal.

- Domain Theory

  A set of facts and rules used by the system to deductively prove that the training example is a member of the goal concept.

- Operationality Criteria

  Criteria that determine when a rule produced from the generalized proof tree is efficient or operational. A rule is operational if it allows the system to perform a task that was intractable before or if it helps the system generate a better solution with less effort. The operationality criteria insures that the system accomplishes learning via the training example.

The training example might seem superfluous because the operational description of the concept is in the deductive closure of the domain knowledge. It is, however, not superfluous because it directs the search through the concept space. Without this guidance, finding an appropriate description can be intractable. Thus, the example, along with the given knowledge, allows the system to determine the correct definition for a class.

2.1.1 EBL Structure Manipulation

The goal concept is provable because the system is given a complete and accurate domain theory. Initially, the learning system is given a positive example. It is known that the example is a member of the concept class and that an example with the same attribute values is also a member of the class. The EBL system proves that the example is a member of the concept class using rules from the domain theory. This results in a proof tree which identifies the attributes (and their values) relevant for the concept class (Figure 2.1). In Figure 2.1, the arrows point from the antecedents to the consequent of the rules. All the rules used here are Horn clauses, but this need not be the case.

The EBL process prunes away the interior of this proof tree when constructing a new rule. This new rule is shown by the `Rule` tree of the figure. The interior portion of the `Built Structure` tree is removed. This can be done without an inductive leap because the domain knowledge is guaranteed to be accurate and complete. Although the figure shows just the leaf nodes of the proof tree being used in the constructed rule, these nodes can be thought of as

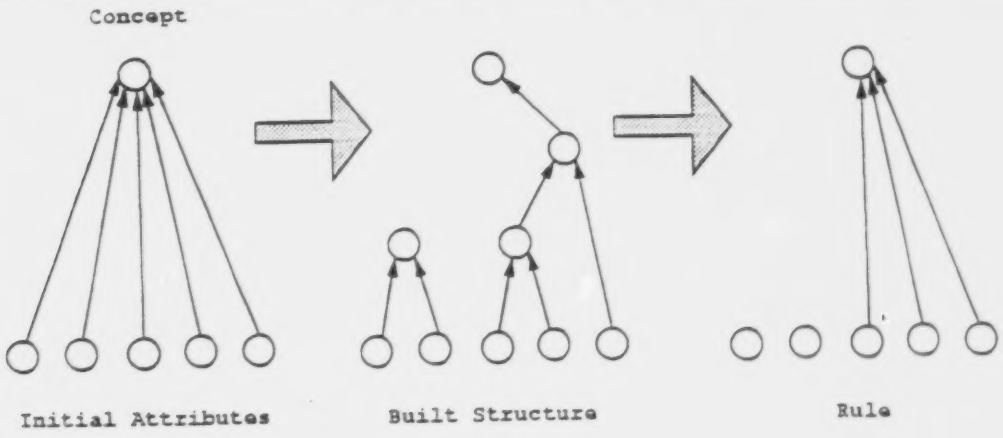


Figure 2.1: EBL Rule making Process

the *operational* nodes of the proof. By reducing (chunking) structure from the proof tree, the learning system has created a new, more efficient rule for concept identification.

2.1.2 EBL Problems

Though one study indicated that people perform EBL [Ahn *et al.*, 1987], it is clear that an intelligent system must also learn at the knowledge level [Dietterich, 1986]. An EBL system cannot create new rules that extend beyond the deductive closure of the current rule set. This is problematic when attempting to use EBL in the construction of knowledge bases for real world problems. One of the main thrusts of machine learning is to assist in knowledge base construction. For EBL to work, however, the complete knowledge base must already be present. Thus EBL learns at the wrong time to be useful as a knowledge acquisition aid. EBL learning occurs after all the knowledge is present in rule form.

Standard EBL systems [Mooney, 1988, O'Rorke, 1987, Segre, 1987, Shavlik, 1988] assume that the system has accurate, as well as complete, domain knowledge. In real problem solving tasks, this is the exception and not the norm. Human knowledge engineers construct the knowledge bases with no guarantee of producing error free knowledge. EBL systems cannot extend rules, even if they are nearly correct, because the repairs are outside the realm of deductive generalization.

Explanation-based learning is an efficient form of learning when all the domain knowledge is gathered. The training example guides the learner to produce useful macro-operators. Such systems are important for making a knowledge-based system more efficient.

2.2 Similarity-Based Learning

Similarity-based learning systems learn new concepts by recognizing patterns of attributes having similar or different values [Quinlan, 1986, Dietterich and Michalski, 1983, Stepp and Michalski, 1986a, Fisher, 1987]. SBL has been successfully applied to learning-from-examples and learning-by-observation tasks using both batch and incremental algorithms[1]. SBL techniques are powerful in the sense that new knowledge can be obtained by the learning system [Dietterich, 1986]. The goals of an SBL learning system are:

- Extend problem-solving capabilities

  The knowledge gained allows previously unsolvable problems to be solved. The performance element improvements are due to better focusing and improved structure of the extended knowledge base. SBL systems generalize the knowledge contained in examples by structuring the knowledge base in a meaningful manner [Lebowitz, 1986a]. This capability assists in achieving the other goals of the system.

- Reduce storage requirements

  Storing a single, generalized rule is more memory efficient than storing multiple examples.

- Reduce problem-solving time

  A new rule can allow the performance element to work more efficiently. Improved efficiency is normally considered a trait of EBL systems, but it is also a trait of SBL systems.

- Directing future focus

  SBL rules specify which attributes are empirically relevant. The rules can direct the performance element to the important attributes of a problem. This relevance information can be used to expedite the problem solving process. Also, it can aid future analogical learning and similarity-based learning.

[1] In learning-from-examples, the system is provided with distinct, classified examples. In learning-from-observation, the examples are not classified and need not be distinct. That is, the examples could be embedded within a stream of observation and the learning system must determine what constitutes an example.

Figure 2.2: SBL Structuring of the Knowledge Base

Figure 2.2 shows how SBL accomplishes some of these goals. Let each e_i be an input event to the learning system. Let each D_i be a decision class. Let each C_i be an intermediate concept class. Assume that the performance element of the system is a case-based reasoner [Kolodner *et al.*, 1985]. By introducing the intermediate concepts, C_1 and C_2 the learning system has simplified the matching process for the reasoner. The reasoner can now work with concepts C_1 and C_2 rather than all the events. If the new intermediate concepts are found to be robust, the reasoner can save memory by forgetting the original examples.

The components of SBL systems are

- Set of examples

  These may be divided into classes by a teacher or the system may have to discover classes (clusters) on its own.

- Learning algorithm that can recognize similarities and differences in the events

  The algorithm manipulates the definition language to produce a more general concept.

- Biases to guide the learning algorithm

  The form of concepts produced by an SBL system are biased by the descriptive power of the representation language used. In addition to language bias, the system has criteria that determines when a concept is acceptable. For example, the biases might prefer conjunctive concepts to disjunctive descriptions. The preferences may be built into the learning algorithm or they may be parameters that are set at system run time.

An SBL system tries to create an empirical rule that describes the given examples. The rule may not be 100% accurate on future examples, even when accurate on all the training examples.

The representation language can limit the search space for the inductive algorithms by eliminating the possibility of certain concept descriptions a priori, e.g., no disjunctive descriptions allowed.

Rules in an SBL system are used for constructive induction; the construction of additional example properties. The rules are used in a forward chaining manner. The system starts with the antecedents (primitive attributes) and concludes higher-level attributes from those features. Most SBL systems do not perform constructive induction in an exhaustive breadth-first manner, applying all applicable rules. Instead, they try some subset of the rules, using heuristics to determine what rules should be fired. An SBL system gains no benefit from backward chaining from the concept to be learned, because no rules in the system that have the concept to be learned as a consequent. The system is trying to construct a rule of the form

$$\textit{Event's common attributes} \Rightarrow \textit{Concept}$$

This is an induced (non-truth preserving) rule, but it is the best the system can produce given the state of its knowledge.

2.2.1 SBL Structure Manipulation

SBL systems work by creating structure. A rule is constructed from observed and inferred (through constructive induction [Matheus, 1989a, Matheus, 1989b, Rendell, 1989]) attributes. SBL finds the common attributes of all the provided events in the desired concept (these are called the training events). This is shown in Figure 2.3. At first, the system can only create rules that are disjunctions of the events seen. Each disjunct of the rule is just a particular instance, as in `Initial Attributes` of the figure. The `Constructive Induction` diagram shows the attributes of a training example, along with the properties that were inferred via constructive induction. In this case two new properties were created through constructive induction. The top node is the concept to be learned. After the system has been presented with many training examples of the concept, it constructs a rule using the most relevant descriptions possible. To produce a characteristic description, the system uses the lowest nodes in the tree that cover the training events. To produce a discriminate description, the system uses the highest nodes in the tree that describe the training events examples but do not cover exampies excluded from the concept class.

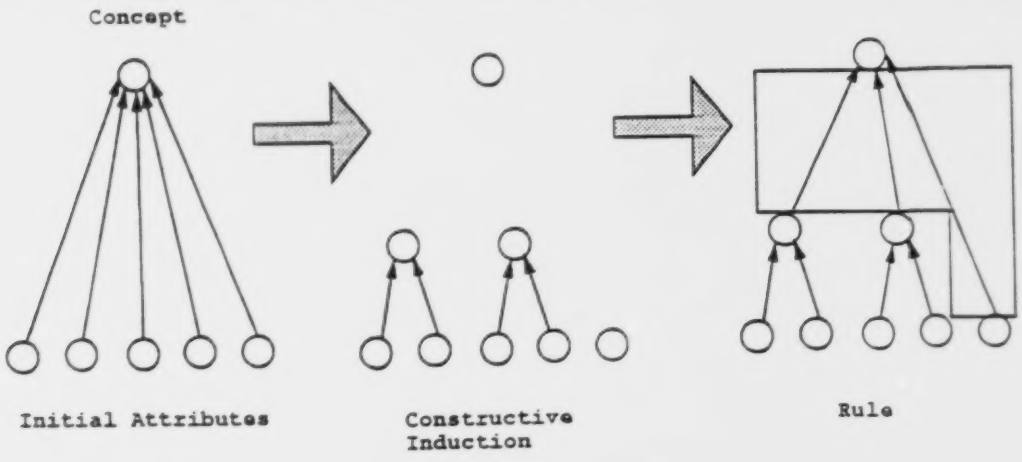


Figure 2.3: SBL Rule making Process

Rules created by SBL introduce a single level into the structure. Many intermediate rules, however, might be needed to explain the complete rule (reasoning) structure. SBL's one level rule is adequate if the problem solver never needs to reason about the hidden subgoals (those intermediate subgoals not produced with the new rule). Some SBL systems can discover the internal complexities of a structure as demonstrated by SUBDUE [Holder, 1988] and PLAND [Whitehall, 1987].

2.2.2 SBL Problems

Though powerful, these systems are easily misled. Even if some attributes are irrelevant to the learning task, those attributes might be used in the learned rule (unless the examples preclude such an interpretation). For example, suppose the system sees John, an adult wearing a blue shirt, successfully move a heavy box. Next, it sees Ed, a child wearing a red shirt, try to move the box and fail. Instead of learning that adults can move heavy boxes, the system might propose that people wearing blue shirts can move heavy boxes . Thus, current SBL systems have no notion of *relevance* when they are searching for differences in examples[2].

[2]SBL systems do use statistical relevance when constructing rules. When limited examples are available, statistical relevance can be misleading.

Because of the problem described above, users of SBL systems must determine which attributes are likely to produce good results. Examples presented to the system then use this limited set of attributes in the induction. The user also must select which examples are relevant for the concept to be learned. The system learns a description for the events provided but only limits the applicability of that rule by the negative events.

Current empirical approaches cannot build up the rich rule structures required for complex problems. The soybean diagnosis system, as described in [Michalski and Chilausky, 1980], was a simple system in which one layer of rules was sufficient. The empirical approach does not work for complicated problems such as constructing process plans for machining complex parts[3]. For empirical systems, the domain expert must control when the learning occurs and provide biases for the system.

2.3 Related Integrated Learning Approaches

Other researchers have recognized the problems of using pure explanation-based learning or pure empirical learning techniques. This has sparked an interested in learning techniques that attempt to compensate for the problems of the previous learning approaches. In this section, some of the other work in this area will be described along with how it differs from the approach taken in knowledge-based learning.

2.3.1 EBL with Theory Revision

There has been some work within the EBL research area on correcting the problems of the more traditional systems. Chien [Chien, 1987, Chien, 1989] has worked on the intractable domain theory problem. Chien's system has complete knowledge but cannot use all of the knowledge due to time constraints. In these situations, the system creates a rule using the standard EBL methodology on a proof tree that is not completely grounded. The system makes assumptions about how the tree will be grounded. When the created rule fails, the failure can be explained because the knowledge in the knowledge base is deeper than the assumption that was made. Another approach used in EBL systems is to perform experiments to determine what

[3]The author has tried constructing knowledge bases for the PHAST system (described later) by using this approach with no success. Even with large training sets the resulting rules were not adequate for solving a single problem not provided during the training. These results are presented in Appendix A.

aspects of the unused domain theory should be included in the constructed rules [Rajamoney, 1988]. These approaches differ from knowledge-based learning because KBL does not have the knowledge in the domain theory to correct the problem.

2.3.2 UNIMEM

Lebowitz [Lebowitz, 1986b] combines SBL and EBL to improve the performance of UNIMEM [Lebowitz, 1986a]. This system uses the rules discovered by the SBL portion of the algorithm as examples to be explained and further generalized by the EBL portion. By giving the generalizations discovered by the SBL system as an EBL example, the EBL system is working with only empirically supported descriptions and is not attempting to work on an intractable number of possibilities. KBL uses the reversed approach, where the knowledge (rules) guide the selection of examples for induction.

2.3.3 CLUSTER/CA

Using an approach similar to UNIMEM's, the CLUSTER/CA system [Mogensen, 1987] extends the SBL system CLUSTER/S [Stepp, 1984] by using a *goal dependency network* (GDN) [Stepp and Michalski, 1986b] to direct the building of classifications. The GDN is used to identify relevant learning goals for the system. With these goals, the system is able to build clusters that more closely match the desires of the user. The GDN can be thought of as a nongeneralizing EBL system, i.e., a simple backward chaining mechanism. PLAND [Whitehall, 1987] integrates SBL and EBL in the same manner but uses multiple levels of knowledge instead of a single level. These approaches use the knowledge to select relevant attributes for learning but use all of the examples provided. KBL uses knowledge to select specific examples during induction.

2.3.4 OCCAM

Pazzani has integrated SBL and EBL methods by making the EBL component guide the process. OCCAM [Pazzani, 1985, Pazzani *et al.*, 1986] is an EBL system that learns to predict and explain the outcome of observed events. OCCAM uses similarities to build a causal explanation when the EBL portion does not contain complete background knowledge [Pazzani, 1987, Pazzani *et al.*, 1987]. In this implementation, the EBL portion focuses the empirical method

on aspects that directly relate to the given situation, thus reducing the attributes given to the SBL system to those that are most significant to the problem. OCCAM's knowledge base is incomplete and can have incorrect rules. The system will correct the rules with errors when an example is presented that invalidates the conclusion of a schema. The system depends upon the user to provide it with examples in an order that allows learning to occur. Pazzani notes that there are many possible ways the system could complete an inference chain, i.e., there are many places where the induction could occur to solve the problem. OCCAM assumes that all the schemas provided for classification are complete. As described later, KBL must compute (possible) intermediate values required to solve the examples it is provided.

2.3.5 GEMINI

Danyluk's system [Danyluk, 1987] uses a simple explanation of a terrorist event to guide the similarity-based frame matching algorithm to build a composite event of the situation. The system first constructs a proof tree using a simple (tractable) database to explain the training event. The features identified by the EBL system are used in the SBL portion to induce causal cues for the event. A different EBL database is then used to explain this induced rule. The second EBL database has rules at a smaller grain size. If the EBL module cannot construct a proof tree, the SBL system is run again to construct a different induced rule. This cycle continues until the EBL portion can explain the induced rule. An important feature of this system is that EBL and SBL methods can call each other recursively to direct the search for knowledge in both directions.

2.3.6 IOE

The IOE (Induction Over Explanations) method of Flann and Dietterich [Flann and Dietterich, 1986] is different than the methods described above. In IOE, the system is given multiple positive examples. A proof structure is created for each of the examples similar to the constructed structure of an EBL algorithm. Using SBL techniques the system combines the proof trees into a single proof for the concept. IOE has been implemented in the WYL2 system. a program for learning chess and checker concepts, such as knight fork. The advantage of such a system is that it removes much of the burden from the person constructing the background knowledge. The IOE methodology allows constants to remain in the solution tree, even if the

background knowledge rules had variables in that position. Thus the proof tree for knight fork has the restriction that the attacking piece be a knight. The KBL approach does not perform induction over proof structures. The goal of a KBL system is to learn rules that will allow a complete proof structure to be constructed.

2.3.7 IOU

The IOU system (Induction Over the Unexplained) [Mooney and Ourston, 1989] specializes a general domain theory. The system requires that the provided domain theory is correct, but does not require that subconcepts be defined. IOU is provided with two classes of events and a single proof structure can be used to explain all the positive events. The proof structure just explains how the examples are part of the general concept. The system then performs induction using the positive events and any negative events covered by the proof. The features covered by the proof are removed before the induction occurs. The system is able to construct rules for the subconcepts of the general concept by adding the inductively found features with the features from the proof structure.

2.3.8 TEIRESIAS

Research into the area of automated knowledge acquisition was started with the work on TEIRESIAS [Davis, 1982]. TEIRESIAS is not a machine learning program, at least by current standards, but the assumptions required to get the system to guide the acquisition are interesting. The knowledge-based learning approach makes many of the same assumptions. First, TEIRESIAS assumes that the control knowledge is separate from the knowledge base. The system is concerned with expanding what the performance element *knows* (domain knowledge), not how it was done (control knowledge). The same idea is used in KBL[4]. TEIRESIAS uses an empirical approach to determine what predicates should be in the premise of a rule. The domain expert can decide if the new rule should be added to the knowledge base. The system looks at other rules in the system (for the head predicate) and uses statistics on predicate use in other rules to construct the rule model. The system assumes that all the rules in the knowl-

[4]In order to get this separation of control from pure knowledge, the original OPS5 rules for the PHAST system were rewritten. The OPS5 rules contained so much control knowledge it is difficult see how a learning system could extend the domain knowledge present. The PHAST system is described in Chapter 4.

edge base are correct. For all the suggestions made by TEIRESIAS about the rules, context plays a very important role. The current problem being solved defines the context in which the added rule should work. Of course TEIRESIAS is not bound by the leaps of faith made in rule construction. The expert approves of all knowledge added to the system. After the knowledge is added the system re-runs the problem to insure that the new rule correctly completes the knowledge base for the problem. KBL uses knowledge of other rules in the knowledge base that match the head predicate to focus attention on attributes. Unlike TEIRESIAS, KBL systems performs induction over a set of examples to to determine the new rules.

2.3.9 DISCIPLE

DISCIPLE [Kodratoff and Tecuci, 1987, Tecuci and Kodratoff, 1989] combines explanation-based learning, analogy, and empirical learning to help complete the current domain theory. The system can work in three different knowledge situations.

1. Complete. When the knowledge base is complete, pure explanation-based learning is used.
2. Weak. This is similar to a pure empirical learning situation. In this situation the system generates examples that it asks the user to classify. The system uses version spaces to keep upper and lower bounds on uncertain rules that were produced.
3. Incomplete. In this situation, there is some domain knowledge but it is not complete. The system performs induction as done in the weak case, but there is more structure to use when selecting the important attributes.

The KBL approach does not require a user to partition the examples; the system performs this task.

2.3.10 ODYSSEUS

ODYSSEUS [Wilkins, 1987] learns when the system is unable to explain the actions taken by the teacher (physician). The system learns rules of a fixed form. It uses the knowledge of the types of repairs to propose all possible repairs. The system prefers repairs that require adding a single piece of knowledge to the explanation already constructed. When there are multiple potential repairs, the system has an ordering on the predicates it prefers. The system uses a

confirmation theory to assist in determining if a rule is correct. The confirmation theory is derived empirically. KBL searches for the best partial proof to complete. and like ODYSSEUS. prefers to make smaller leaps of faith and induce as few rules as possible to solve the problem. KBL does not have the correct conclusion for its rules provided (it is induced), where the physician has provided this knowledge to ODYSSEUS.

2.3.11 Learning from Plausible Explanations

Fawcett's work [Fawcett, 1989] is similar in many ways to the approach taken in KBL. Fawcett's learning extends the unproven nodes of a partial-proof tree. The system uses heuristics to determine which partial-proof tree needs to be extended. Maximally specific descriptions are used to construct rules for the ungrounded predicates. There are a number of differences between KBL and this system. First, Fawcett's system will not use a partial-proof tree that has an ungrounded goal. if that goal is grounded in another partial-proof tree. Using appropriate values for the *factors* of the RSI module (Chapter 5), KBL could work on such a proof tree. Second, the system does not try to generalize the rules created as the information from one problem is incorporated into the rule structure. Instead, the system builds new rules for each example and then later must use some (unspecified) method to combine the rules. The generalization process uses positive examples only.

2.3.12 META-DENDRAL

META-DENDRAL [Buchanan *et al.*, 1976, Buchanan and Feigenbaum, 1978] is a program that learns cleavage rules for explaining the operation of a mass spectrometer. Three modules make up META-DENDRAL: INTSUM, RULEGEN, and RULEMOD. INTSUM uses a half-order theory of mass spectrometry to convert training instances of the form

whole molecular structure ⇒ mass spectrum

into cleavage rules of the form:

whole molecular structure ⇒ one designated broken bond.

RULEGEN and RULEMOD use these cleavage rules to find a small set of generalized cleavage rules that cover most of the training examples. RULEGEN constructs a model-driven generate-and-test search of the rule space. The search begins with the most general cleavage rules.

RULEMOD cleans up the rule set found by RULEGEN. META-DENDRAL uses both examples and domain knowledge to learn cleavage rules.

2.3.13 LEX

LEX [Mitchell *et al.*, 1983] is an interesting program that has some of the aspects of a knowledge-based learning system. The LEX system learns heuristics about when to apply operators it has for performing integration. The main feature that LEX has in common with KBL systems is that it collects examples. LEX considers operators on the path to the solution to be positive examples of operator application and operators not on the solution path are negative examples. Thus the system determines what are positive and negative examples from a context, albeit a simple context. But LEX does not induce (create) the rules or operators. That is, it does not learn new consequences for the rules. These are all provided initially. The system just learns antecedents for the production rules.

2.3.14 Relation of Other Work to KBL

Each of the previous systems combines an empirical algorithm with a deductive method. But none of the systems attempts to integrate the processes. At some level there must be a distinction between the deductive and inductive processes, but with these systems a large grain size has been defined for the separation of the methods. The knowledge-based learning defines a much smaller grain size between the processes, allowing them to interact more tightly. None of the related systems do all of the functions performed by KBL:

- Select examples for the induction without user intervention.
- Determine where knowledge is missing and induce both the consequent (binding for the missing variable value) and the antecedents of the new rule.
- Partition events into classes to avoid over generalization of the new rules.
- Handle real numbers and discover mathematical formulas.

Chapter 3

Knowledge-Based Learning Paradigm

One of the goals of machine learning is to construct knowledge bases for expert systems. Likewise, a goal of this work is to aid in the construction and maintenance of expert systems by assisting a knowledge engineer in the formulation of declarative knowledge. This research focuses on learning knowledge missing from the knowledge base. It also shows how knowledge can be used to correct rules. Missing knowledge prohibits the performance element from solving problems that should be within its scope[1]. Knowledge bases are typically incomplete in real world situations. Currently, human knowledge engineers construct knowledge bases and humans are prone to error. Even if knowledge engineers initially construct a complete knowledge base for a domain, the domain will change and the effort and time that originally went into constructing the expert system will be lost.

Another goal of this research is to develop a machine learning algorithm useful in engineering domains. Previous machine learning systems have only limited use in engineering domains because of the stringent requirements they put on the knowledge used in the learning process. Analytical approaches (e.g., EBL) to learning require a complete, and accurate knowledge base. Empirical approaches (e.g. SBL) require that the problem space be well represented with a large set of examples, and that the examples are described with the attributes most likely to

[1]This research is not concerned with the *knowledge cliff* [Feigenbaum, 1984] or *brittleness* [Holland, 1975] problem. Missing knowledge is knowledge that should be in the knowledge base, but is not.

be relevant (as determined by the user). In contrast, the knowledge-based learning approach uses knowledge in the form of rules and solved examples. With this knowledge, a KBL system can learn new rules that extend the problem-solving capabilities of a performance element[2] by focusing the learning component on specific areas of the knowledge base.

Engineers use many forms of knowledge when solving problems. Two of the more important types of knowledge are scientific laws/principles and experience. Engineers are trained extensively in school in the science-based problem-solving techniques important to their field. Learning from textbooks, however, is not enough; it takes years for an engineer fresh out of school to acquire enough experience to become a skilled professional. During this apprenticeship, the novice is exposed to many problems. Eventually, the engineer matures. And although an experienced engineer might never see the exact problem twice, he or she is able to use past experiences to solve new problems. Thus, both science-based knowledge and practice-driven experience play an important role in engineering problem solving. Knowledge-based learning takes advantage of these two forms of knowledge. An engineer formalizes a set of rules that solve many problems. Some problem areas, however, will be overlooked, or the engineer could not translate the knowledge into rules. So, the initial knowledge base is incomplete. In addition to the rules, the engineer provides examples of solved problems. These examples can contain knowledge missing from the rule base. This set of examples will grow over time as the engineer solves more problems.

The next section provides details on how knowledge-based learning differs from other forms of knowledge acquisition. This includes a discussion of what is required of a user of various learning systems. One of the goals of KBL is to remove much of the operational burden from the user. The second section provides a comprehensive view of the KBL methodology. The important and new aspects of the paradigm are coded, tested and evaluated (Chapters 4, 5, 6). The third section explains a methodology for evaluating KBL against other learning approaches. This is followed by some concluding remarks on the knowledge-based learning paradigm.

[2]The performance element of a KBL system is the expert system consisting of the inference engine and the original knowledge base.

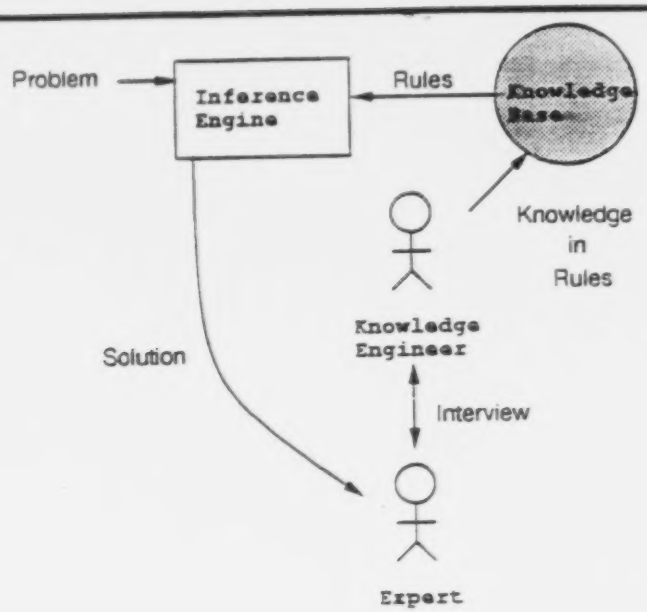


Figure 3.1: Knowledge Acquisition with a Knowledge Engineer

3.1 KA, ML, and KBL

Knowledge acquisition (KA), machine learning (ML), and knowledge-based learning (KBL) are all concerned with constructing a knowledge base for a performance element. Knowledge acquisition is human intensive but is currently the only practiced approach to constructing large, complex knowledge bases [Buchanan *et al.*, 1983]. As Figure 3.1 shows, this process involves two people. The expert has the knowledge that the performance element needs. The knowledge engineer extracts knowledge from the expert into a representation used by the inference engine. Knowledge acquisition is a complex, iterative process. As the first step the knowledge engineer interviews the domain expert extracting knowledge. Next, the knowledge engineer transforms the expert's knowledge and places it into the knowledge base. The knowledge engineer then tests the knowledge with an inference engine. The results of the tests are presented to the expert who identifies errors and shortcomings in the knowledge base. This process then repeats until the domain expert (or final system user) is satisfied with the performance of the system.

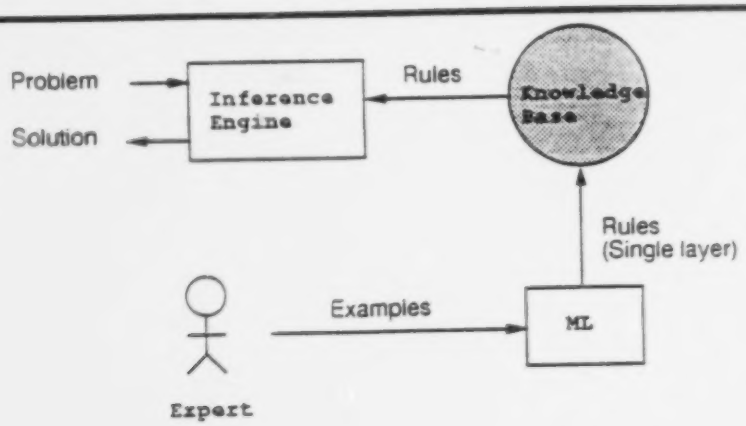


Figure 3.2: Empirical Machine-Learning Approach to Knowledge-Base Construction

One of the goals of ML is to facilitate the knowledge engineering task by replacing the knowledge engineer with an algorithm that constructs the knowledge base using examples supplied by the expert. Empirical machine learning systems use this approach (Figure 3.2). (Analytical machine learning systems cannot be used to construct an initial knowledge base because they have no knowledge to compile.)

Knowledge-based learning takes a more moderate position than pure empirical approaches. KBL interacts with the inference engine in addition to using examples and already collected knowledge base rules (Figure 3.3). Both the knowledge-acquisition approach and the KBL approach to knowledge-base construction depend upon the use of the inference engine. In traditional knowledge acquisition (Figure 3.1), the domain expert critiques the results of the inference engine. In KBL, the inference engine indicates that learning needs to occur by providing the learning system with a problem that it cannot solve. As described in the next section, KBL uses the current unsolved problem to direct the learning process. The system uses examples provided by the expert and knowledge in the form of rules. The rule knowledge can be obtained through traditional knowledge acquisition techniques or by expert consultation with an automated knowledge acquisition system, such as TEIRESIAS [Davis, 1982].

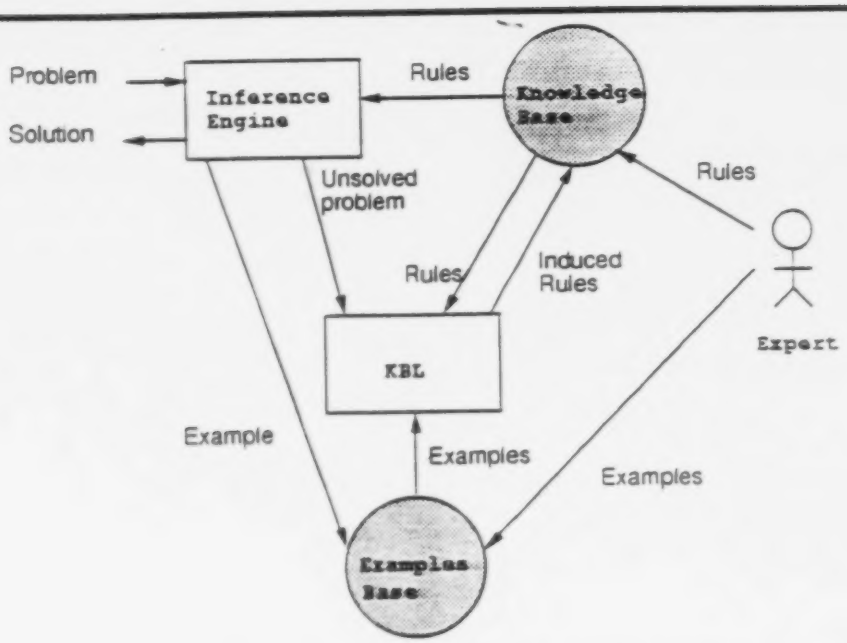


Figure 3.3: Knowledge-Based Learning Approach to Knowledge-Base Construction

3.2 KBL Overview

This section describes the complete knowledge-based learning paradigm. KBL uses knowledge to correct existing rules and induce missing rules from the knowledge base. Figure 3.4 shows the interactions between the main components of a knowledge-based learning system. As the figure shows the KBL process starts with an unsolved problem. The problem is a set of ground facts and an uninstanciated top-level goal. The inference engine is unable to solve the problem and passes it to the learner. The learner first calls the partial-proof tree constructor module to construct a proof tree that provides a possible solution. Because in general, a complete proof cannot be constructed, the induction module will be called to create or repair a rule. The interaction between these modules is actually more complex than depicted in this figure because more than one rule might need to be induced for a complete proof to be constructed. These details will be explained later.

The induction module repairs an existing rule or induces a new one and modifies the knowledge base to reflect the change. The induction module induces new rules using an example base which contains examples of problems and their solutions. The partial-proof tree constructor

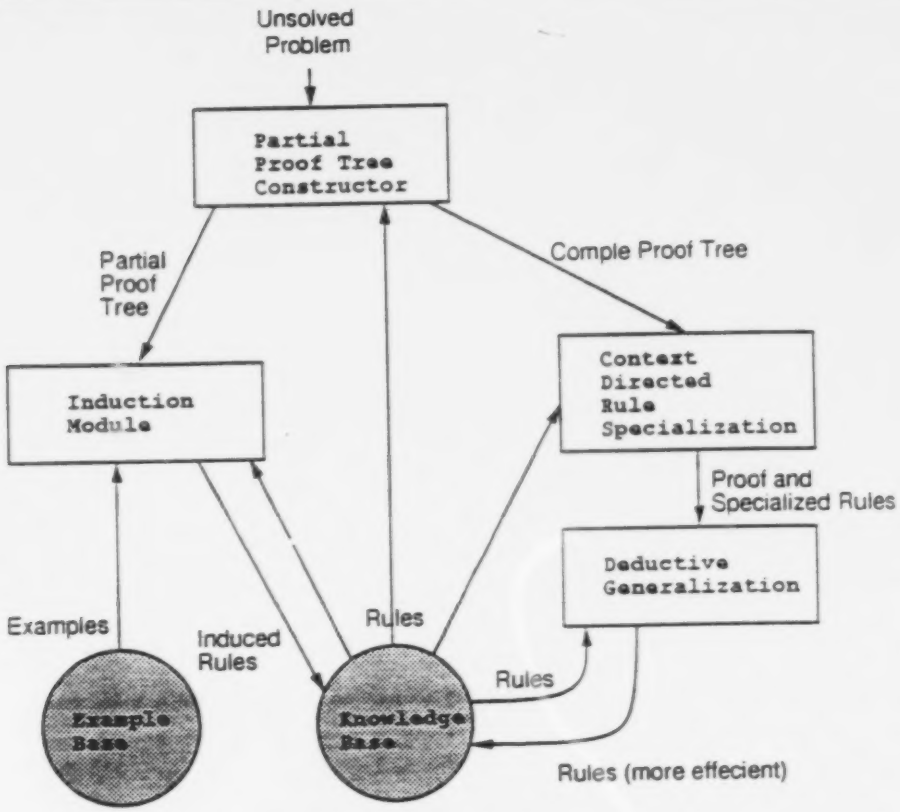


Figure 3.4: Details of KBL Approach to Knowledge-Base Construction

can then continue until a complete proof is built. Since rules could be induced to allow many partial-proof trees to complete, if desired, multiple solutions can be computed. If multiple proofs with conflicting answers are constructed, the system must correct some of the rules used in the proofs. The conflict directed rule specialization module does this. When the system has a single proof, it should determine if deductive generalization should occur. This is an EBL process that will allow the system to solve similar problems more efficiently when they are encountered in the future.

KBL is initially provided with an example base and a knowledge base. The example base consists of facts for a problem (what is given to the performance element) paired with a solution. The expert provides the example base initially, but solved problems are added as the system runs. Problems solved by the system can be used to assist in generalizing and specializing

induced rules. The initial knowledge base is assumed incomplete. The goal of the system is convert knowledge from examples to rule form.

The remainder of this section describes the components of KBL in more detail. Implemented versions of these components are discussed in Chapter 4 and Chapter 5.

3.2.1 Partial-Proof-Tree Constructor

Constructing a proof tree from an incomplete knowledge base is more complex than backward chaining because some subgoals do not have rules. A failed subgoal does not necessarily mean that the rule is inappropriate for this situation and that backtracking should occur. Instead, it could identify a gap in the knowledge base. Because the system cannot know how to interpret a failure, the construction of partial-proof trees relies on heuristics described in the next paragraph). The KBL system can never be certain that the partial-proof tree that is passed to the induction module correctly identifies a place where knowledge is missing.

Here are the heuristics used to guide the search:

1. Perform as little induction as possible. The system should find solutions using the existing rules first.

2. Prefer inducing rules for more complete proof structures. This heuristic is important for two reasons. First, more complete partial-proof trees provide a more detailed context for rule induction. Second, by assumption, problems in the rules used for the earlier portions of proofs have been corrected, if the proof can proceed to later subgoals.

3. Modify existing rules to complete a proof before inducing new rules. Of course, the system should avoid over generalizing an existing rule.

4. Solve subgoals that can be grounded before passing the partial-proof tree to the induction module. This can be done by checking that all the remaining subgoals can be solved or by assuming that the subgoals are ordered. The ordering would insure that subgoals occur later in the rule (to the right of) than the subgoals on which depend.

The module used to search for proofs may return more than one proof. This is useful if the proof provided by the system is incorrect (as detected by an oracle) or if more than one proof can be constructed and if each proof gives a different answer (and no induction occurs to complete

the proofs). As explained later, these proof trees can be used to correct overgeneralized rules in the knowledge base.

If rules must be induced, the search for complete proofs is complicated. The system could induce a rule at many different levels in the proof structure. The position of the unsolved subgoal in the structure will help guide the induction module and effect the search for other proofs.

3.2.2 Induction Module

The induction module for knowledge-based learning systems is more complex than traditional empirical learning methods. The KBL induction removes much burden from the user. With traditional empirical learning systems, the user must supply the relevant examples with the relevant attributes to the system. The KBL induction module selects the relevant examples based upon the partial-proof tree and uses the knowledge base to focus attention on the attributes most likely to be important to the problem. Also, due to the representation required to solve engineering problems, the system must discover the consequent of the rule in addition to inducing the antecedents. Figure 3.5 shows how the induction module works. The following sections describe each component.

3.2.2.1 Gather Examples

A KBL system tries to select relevant examples from an example base instead of depending upon the user to provide them. Ideally, the examples collected define the missing subgoal precisely so that induction builds the correct rule. The examples are selected (by the KBL system) based upon the *context* of the current problem. The context is the portion of the partial-proof tree that has been solved (all variables in the leaves are grounded). The ungrounded version of the context makes up the constraints. The system applies those constraints to the example base and collects the examples that meet them. Figure 3.6 contains a simple partial-proof tree. The context from that PPT is `(a1 ?y) (a2 ?y) (b1 ?z)`. KBL assumes that a solution for an example matching the context is similar to the solution needed to correct the problem.

Throughout this work, the ungrounded version of the leaves of the partial-proof tree will be used to define the context (e.g., `(a1 ?y) (a2 ?y) (b1 ?z)`). This need not be the case.

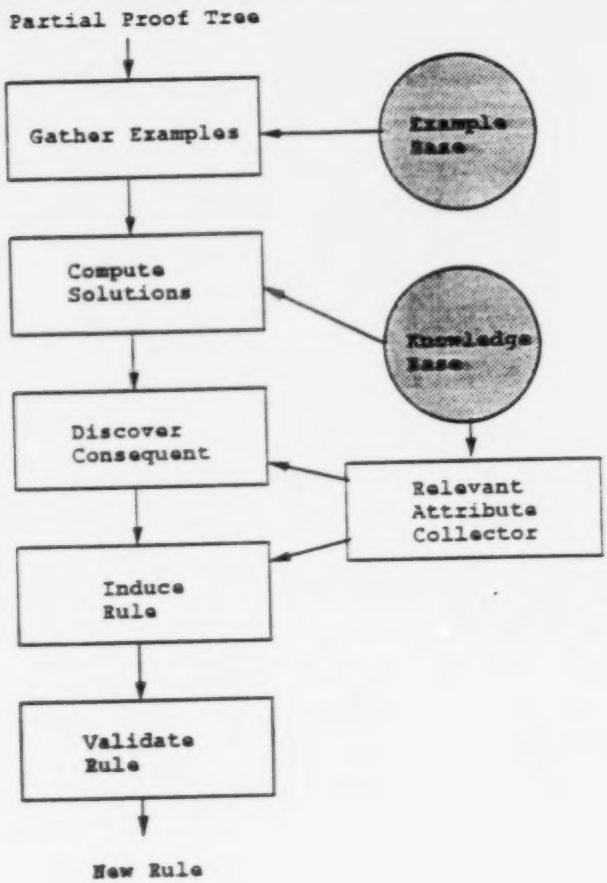


Figure 3.5: Knowledge-Based Learning Induction Module

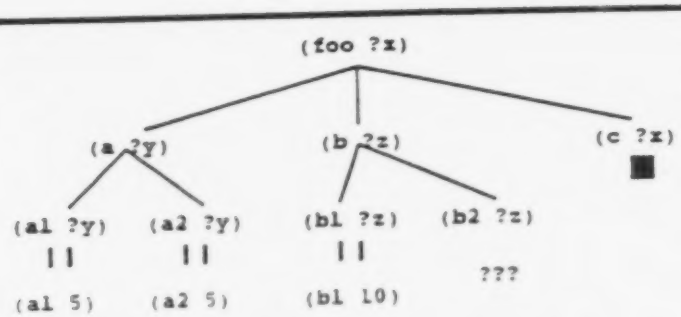


Figure 3.6: Example Partial-Proof Tree

If the system was provided with metaknowledge about the relationships of predicates then more intelligent context pruning could occur. Prunning involves ignoring those portions of the partial-proof tree that do not effect the portion needing a new rule. This idea is similar to defining operationality in EBL systems [Keller, 1988]. For example, if the system had additional knowledge indicating that the details of finding a solution for a are unimportant to b, the context would become `(a ?y) (b1 ?z)`.

3.2.2.2 Computing Solutions

The system now has a set of example/solution pairs from the example base that match the context. To perform induction, the examples must be classified. Before they can be classified the value of the unsolved subgoal must be determined. In the example of Figure 3.6, the system is trying to induce a rule for b2, as indicated by ???. The set of examples collected have solutions for `foo` — the top level goal. The induction module must derive from the value for `foo` a value for b2 so that the value can be used to assign class membership to the examples. As will be described in Chapter 5, in general, the best the system can do is constrain the solution to a range of possible values. Fortunately this method is adequate to allow induction to continue.

To get the set of values to which the missing variable can be bound, the system must work backwards from the solution to the point in the tree where the variable is bound. The first place where the variable is used (traversing the tree in pre-order) indicates where it is assigned a new value. References to the variable later in the proof tree are only accessing the bound value of the variable. These references are used to constrain the actual values that would allow a proof of the problem to be solved.

3.2.2.3 Relevant Attributes Collector

Real-world problems can have many attributes. For example, the traffic problem described in Chapter 4 has over 100 attributes. The system must, therefore, focus on the attributes that are relevant to the unsolved goal. In traditional empirical learning, systems use all the attributes (or the user selects the relevant attributes). Focusing on relevant attributes helps the system produce sensible results. In addition, it helps reduce the problem space. As more attributes are used in the induction of a new rule, more examples need to be used to identify areas outside the scope of the new rule (i.e., more negative examples are required).

Relevant attributes can be identified in at least two ways. Knowledge can be provided indicating which attributes are more likely to be useful when inducing rules for a predicate. Alternatively, the system can look for other rules with the same predicate in their consequent as the subgoal under consideration. The system can focus on the attributes used by these other rules.

3.2.2.4 Discover Consequent

After the system computes the solution to the unsolved goal of each example (or a range of values that bound the solution), the consequent of the rules must be determined. If the missing variable could only be assigned a single, constant value the task would be easy. The events would be divided into one class for each value seen. At the worst, this creates overly specific rules. Engineering problems, however, require a more expressive language. One that allows variables to be bound to the values of other variables or mathematical formulas. The induction module must discover the relationship between the missing values and the other variables. The system must search for the proper expression to place in the rule consequent: constant, variable, or mathematical formula. Not all of the examples need to be covered by one expression. Sets of examples are formed (partitioned) based on the different expressions covering them. The examples for one partition can, later, act as near misses for the other classes during induction. The search used for the expressions relating examples is similar to that used in scientific discovery systems [Falkenhainer and Michalski, 1986, Langley *et al.*, 1987]. The main difference is that for complex domains the system has a bias of finding simpler formulas that cover some subset of the examples over finding one complex formula that covers all the examples.

3.2.2.5 Induce Rule

With partitioned examples and a known consequent for the rule, the system is ready to induce the antecedents for the rule. A traditional empirical learning from examples system can be used to induce the appropriate conditions for firing a rule. The induction system should be biased to produce overly specific rules instead of overly general rules. The system can detect overly specific rules when constructing the partial-proof tree and generalize those rules. In

order to detect overly general rules, an oracle must indicate an incorrect result or the system must search for contradictions (as will be explained shortly).

3.2.2.6 Validate Rule

The induction module has now produced a rule that when added to the knowledge base will allow the PPT constructor to continue. Checking the validity of the produced rule could occur at this time. To validate the rule, a proof is completed for all the examples the rule covers. The solution from the proof is compared with the correct solution for the example. A count is assigned to the rule indicating how many examples matched the results provided. As the number of correctly covered examples grows in relation to the number of incorrectly covered examples, confidence in the rule increases. There are a couple of difficulties with this approach. First, constructing a proof for an example is time consuming. When a rule covers many examples, it is difficult to determine how many to test. Also, the system is not guaranteed to complete a proof for these examples. There could be other rules missing from the knowledge base that are needed to solve the problem. These missing rules should not influence the confidence in the new rule. With the new rule the system can proceed to learn about those other rules for the current problem or future problems.

3.2.3 Context Directed Rule Specialization

The context directed rule specialization module is called after a complete proof (solution) has been constructed. It is possible that more than one potential proof was constructed. If each potential proof computes a different solution, one must be in error. How could this happen? The system induced an overly general that fires in situations where it does not (should not) apply. Until now, only generalization has been discussed as a way of improving a knowledge base. But specialization also plays an important role in correcting knowledge. The next three subsections describe different situations where overgeneralization can be recognized. The conflict-directed rule specialization module handles one of these cases, rule–rule conflicts. There are only three types of knowledge conflict that can occur in a knowledge-based learning system. There may exist conflicts within the knowledge base, conflicts within the example base, or between the two forms of knowledge. Presented next is a brief discussion of these knowledge inconsistencies and the processing that can be done to alleviate the situation.

3.2.3.1 Rule–Rule Conflicts

Rule–rule conflicts occur when multiple, differing solutions can be produced for a single set of problem facts. This situation arises because at least one rule is too general and fires in an inappropriate circumstance. The system cannot determine which solution is correct, an oracle must do so. The system, however, can look back through the proofs constructed and specialize induced rules that have the lowest levels of confidence. Identifying the exact rule in error is difficult. The system can determine where two proofs diverge but the divergence could be anywhere after an inappropriate rule use. The system could also work backwards from the solution to narrow the search to only those rules effecting the incorrect values of the solution.

3.2.3.2 Example–Example Conflicts

Example–example conflict occurs when two identical examples have different solutions associated with them. In the literature on empirical learning, the examples in error are called *noise* [Quinlan, 1986]. Some empirical learning systems can deal with noise in the examples. Such techniques are not addressed in this work and it is assumed that there is no noise in the examples. Techniques developed to handle noise in other machine learning algorithms could be applied to KBL systems [Angluin and Laird, 1988, Schlimmer and Granger, 1986].

3.2.3.3 Rule–Example Conflicts

Rule–example conflict occurs when the solution produced by the knowledge-base rules differs from the example's (provided) solution. Either the example or the rules could be erroneous, but the examples are assumed correct since the system has added induced rules (which may be in error). Therefore, when an incorrect solution is computed for an example, the knowledge base is in error. The error could occur for a number of reasons. A rule might be overly general and applied in the wrong context. The discovered consequent for a rule could be incorrect. One method for the system to correct rules is to have a background process constructing solutions for examples in the example base. When incorrect results are found, the system can try to identify the rule in error and correct it. This involves search because many possible rule modifications could be made to adjust the proof tree. The system would start by adjusting rules that are newer or have low confidence values.

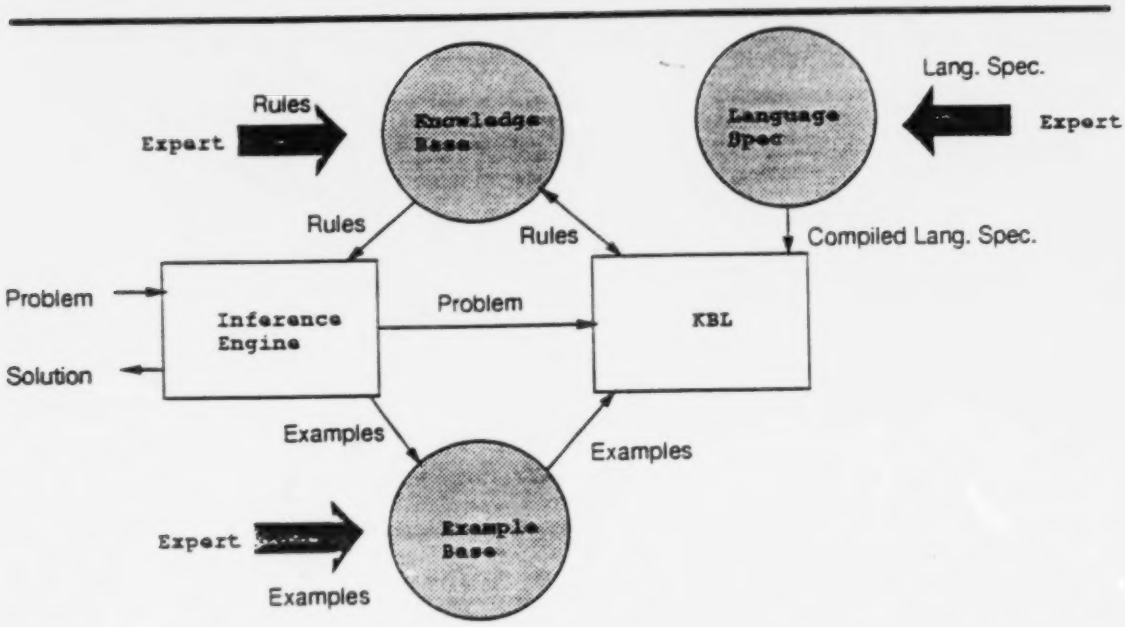


Figure 3.7: KBL and Performance Element Interactions

3.2.4 Deductive Generalization

Induction is not the only way in which the performance of a knowledge based system can be improved. The system has a single solution for the problem at this point. The system now determines if the performance of the knowledge base is adequate for the user's needs. If not, deductive generalization (EBL) compiles knowledge into the rule base. If the performance is adequate, as specified by parameter, no compilation of knowledge occurs.

3.2.5 KBL Overall Structure

Figure 3.7 shows the interactions of the performance element, knowledge base, example base, and language specification with KBL. These are the required inputs to the KBL:

- Incomplete domain knowledge.
- Set of examples with solutions (example base). Examples are added from the performance element as problems are solved.
- Language specification.

- A problem that cannot be solved with the incomplete domain knowledge.

There are a number of constraints a real world system would want to obey. First, the user of the system should be able to constrain the amount of time the system spends searching for a proof. The user should also be able to limit the amount of time allotted to the induction component of the system. The system should take care that rules added to the knowledge base do not adversely effect the efficiency of the system [Minton *et al.*, 1987, Holder, 1990]. A more advanced KBL system should be able to work with probabilistic rules. (Constructing proofs with such rules is a research topic at this time [DeJong, 1990].)

3.3 Methods for Evaluating KBL

If machine learning research is to be good science, it must be evaluated. This section describes methods of evaluating the performance of a knowledge-based learning system. The principle problem in measuring the benefit of using a KBL system is determining how the domain knowledge influences the construction of rules.

The following factors determine how well a KBL system learns:

- **Effectiveness**

  Effectiveness of an algorithm is determined by how often rules must be learned for a data set. If the system is learning rules effectively, i.e. making the appropriate generalizations, the learning module should be required less frequently than if the learning system is learning ineffective rules. If new rules are not learned frequently that means one of two things. One situation could be that the test examples might not require the use of knowledge missing in the knowledge base. In the second case, it is assumed that the testing examples do require knowledge not initially in the knowledge base. The system learned rules at the correct level when presented with problems, otherwise errors would occur or many new rules would have to be learned.

- **Correctness**

  KBL should produce rules that improve the accuracy of the knowledge base when solving problems. Accuracy can be measured as a binary notion (correct/incorrect) or with some established criteria for a domain that indicates an answer is close to the correct answer. It

is important that as new rules are introduced into the system, previously solved problems do not become unsolvable or incorrectly solved.

- **Resourcefulness**

  One of the beneficial aspects of KBL is the ability to reduce the number of examples and attributes that are considered during the induction phase. Resourcefulness is a measure of how well the KBL system selects examples and attributes for the learning process. The system wants to select the most appropriate examples and attributes but ignore irrelevant ones. At the same time, important events and features should not be left out of the learning process. Measuring the resourcefulness of a system is subjective because it is not always possible to objectively determine the quality of the new rules produced.

- **Understandability**

  KBL systems should produce rules that the domain expert can understand. Clearly, this is beneficial if the performance element explains the results acquired by tracing through the used rules. In addition, understandable rules can be presented to a domain expert before being added to the knowledge base. The domain expert could determine if the rule makes sense or if something critical is missing. Objectively determining the understandability of a rule is problematic and will not be addressed further.

This section describes testing methods that can be used to evaluate a KBL algorithm. In measuring the effectiveness and correctness of a KBL system, each component must be dealt with separately. Understanding how to compare two KBL systems helps to understand how to compare KBL systems with other types of learning systems.

If comparing two KBL systems, first the accuracy of the results would be compared. If the accuracy (correctness) is the same for both systems, one would chose the system that is most effective. The system with the smallest learning overhead and highest accuracy is preferred. Testing the accuracy in these two systems is simply a matter of providing each KBL system with an initial knowledge base and example base. Then, a set of training examples would be run on each system so that new rules would be produced. The systems could be tested with a new set of examples that only required knowledge that did not exist in the initial data base. There is no need to run examples that could be solved with the a priori knowledge base[2]. The

[2]This assumes KBL systems, like KBLØ that do not modify existing rules in the knowledge base.

system that solved the most testing examples correctly would have the better accuracy. If the systems solved the same number of examples, the number of times the inductive portion of the learning algorithm was called could be counted to determine the more efficient algorithm.

When comparing two KBL systems, each system can be provided with the same initial knowledge. When making evaluations between KBL and non-KBL systems, one must consider the two components of the KBL system. The initial knowledge provided to the KBL system, in the form of rules, must be considered. The learned knowledge enables the system to perform beyond its previous capabilities. If the KBL system is provided with a nearly complete knowledge base, and a set of training examples randomly selected from the natural distribution of all possible examples, there might not be any training examples that cause the KBL system to learn. All the training events might be covered by the knowledge base. Even though learning never occurred, the system could have good accuracy. Comparing this KBL system with an empirical system that must learn all the rules will make the KBL system appear better. Such a test only determines if the initial knowledge base or the empirically learned descriptions are better.

3.3.1 Evaluating Effectiveness

The effectiveness of a KBL system is determined by how well training examples and example base knowledge where converted into rules. If the deductive closure of the knowledge base extends past problems that should be covered after a specific training example, the system is effective. Thus, how is one to judge if a KBL system learned too many rules or not? The examples presented to the system might all be unique or could all be the same. To judge if the system is effective in its learning strategies there must exist a standard against which KBL results can be compared. Determining an appropriate standard is difficult. Explanation-based learning systems are not appropriate because no learning occurs if the knowledge base is incomplete[4]. That means empirical approaches to learning are left as a standard. Using these systems presents a number of problems. First, empirical approaches do not use any sort of knowledge base, only the examples are used. For example, if ID3 [Quinlan, 1986] is used to construct a decision tree based upon all the examples in the KBL system's example base, the

[4]Learning can occur with portions of the knowledge base that are complete.

decision tree has different knowledge than what is in the example base. In a sense, the decision tree is complete because it possesses all the knowledge that the knowledge base needs to be completed (using the examples). The knowledge base for the KBL system is incomplete and the examples are there to supplement the knowledge base. Thus, initially the KBL system is placed at a *knowledge disadvantage.*

Realizing the problems stated above, there are some methods for determining how effective a knowledge-based learning system is in the construction of new rules. For each of the tests described below, the KBL system is compared to a standard empirical learning system, such as ID3.

1. Determining total errors:

   Present each system with a testing set of examples. Count the number of examples that are processed incorrectly. For ID3, this means the event was classified incorrectly by the decision tree. For a KBL system, the problem could not be solved correctly with the current knowledge base. When there is an error, the systems must learn either a new decision tree or another rule for the knowledge base.

2. Determining mean time between failures:

   Present each system with a testing set of examples. Keep track of the number of examples solved correctly between the times learning must occur. The system with the largest mean time would appear to be using the learning phase more effectively.

3.3.2 Evaluating Correctness

Correctness is a measure of accuracy of the knowledge learned. Learning curves are typically used to measure the correctness of a learning algorithm. As more examples are presented to an empirical system, it should better predict unseen examples. Such tests often tell the observer more about the complexity of the domain than the correctness of the learning algorithm. More complex domains require more examples to get good results. Comparing two empirical learning systems is useful because relative differences are highlighted. Making such comparisons between KBL systems and empirical algorithms is difficult for the same reasons stated in the previous section.

As an example to make this clear, let ID3 be run on a training set. This same training set is used as the example base for a KBL system. The first testing example is presented to both systems. If ID3 classifies the testing example correctly, one has more confidence in the decision tree. Present the same example to the problem solver of a KBL system. If the system solves it correctly, nothing is learned. There must be a problem solving failure in order for a KBL system to learn. So is it fair to say the KBL system is correct? The knowledge base is accurate for the example presented to it. If the performance element had failed to solve the problem, learning would have occurred and the KBL system would *then* use the examples in the training set. The training set is used at different times for each system.

To test a KBL system there needs to be (at least) three sets of examples. One, the set of examples used in the example base of the system. The second set is a set of examples used to train the KBL system. Examples that are used to extend the current knowledge base. The third set is used to determine the actual correctness of the system. The training set for an empirical learning system consists of the first and second sets used by the KBL system.

3.4 Summary

The knowledge-based learning approach is motivated by the observation that empirical and explanation-based learning systems do not perform well during knowledge-base development. EBL systems [Mitchell *et al.*, 1986, DeJong and Mooney, 1986] improve the performance of a knowledge base once all the rules are known. Empirical learning approaches [Quinlan, 1986, Dietterich and Michalski, 1983, Stepp and Michalski, 1986a] construct decision rules that work well for an isolated part of the rule base, but these systems do not produce rules that merge with a partially developed knowledge base. Knowledge-based learning addresses the problem of learning when some knowledge already exists in rule form, and some other knowledge exists as solved examples for the domain. KBL systems determine which solved examples are relevant to extending the deductive closure of the current knowledge base.

There are four important reasons for developing knowledge-based learning systems:

- Non-KBL learning systems produce rules that do not install into an existing knowledge base.

- Non-KBL learning systems require the user to select the examples and the attributes to be processed.
- Non-KBL learning systems require the user to control when learning occurs.
- Non-KBL learning systems do not repair the incompleteness of knowledge bases using empirical approaches.

Knowledge-based learning is a control mechanism that frees the user from the burden of judiciously selecting examples for the learning algorithm and fitting the new or revised rules into the knowledge base. In order to accomplish these goals the system is coupled with the performance element. KBL systems require the same information that is required of both EBL and SBL systems, examples, domain knowledge, language specification, and a problem. Evaluating KBL systems with non-KBL systems is difficult because examples are used differently in the systems and KBL is extending existing knowledge. One must measure the improvement in knowledge.

Chapter 4

KBLØ

Knowledge-based learning systems work at the request of a performance element. When the knowledge base is lacking knowledge that would allow the performance element to solve the current problem, the KBL system is invoked. The performance element passes the unsolved problem to the KBL system. The KBL system uses the same knowledge base accessed by the performance element to construct a partial-proof tree for the problem[1]. The partial-proof tree defines a context used to select relevant examples for the induction of new rules. The induced rules help complete the knowledge base. Specifically a rule is induced for the missing subgoal in the current partial-proof tree.

The relationships between the performance element, the knowledge, and the KBL system are shown in Figure 3.7. When the knowledge base contains sufficient knowledge for the performance element to solve the problem, a solution is returned and the problem description and solution are stored in the example base. The example base contains knowledge stored in the form of problem description/solution pairs. Examples are added as the performance element solves problems. An expert can also directly store problem/solution pairs into the example base. The knowledge base is a set of rules that compute solutions to problems for the domain. The initial knowledge base could be constructed by traditional knowledge acquisition techniques that resulted in an incomplete knowledge base.

This chapter describes KBLØ, an implemented KBL system. The first section explains the details of how the KBL box of Figure 3.7 was implement for KBLØ. This will lead into a

[1] If a complete proof could be constructed then the problem would have been solved and KBLØ would not have been called.

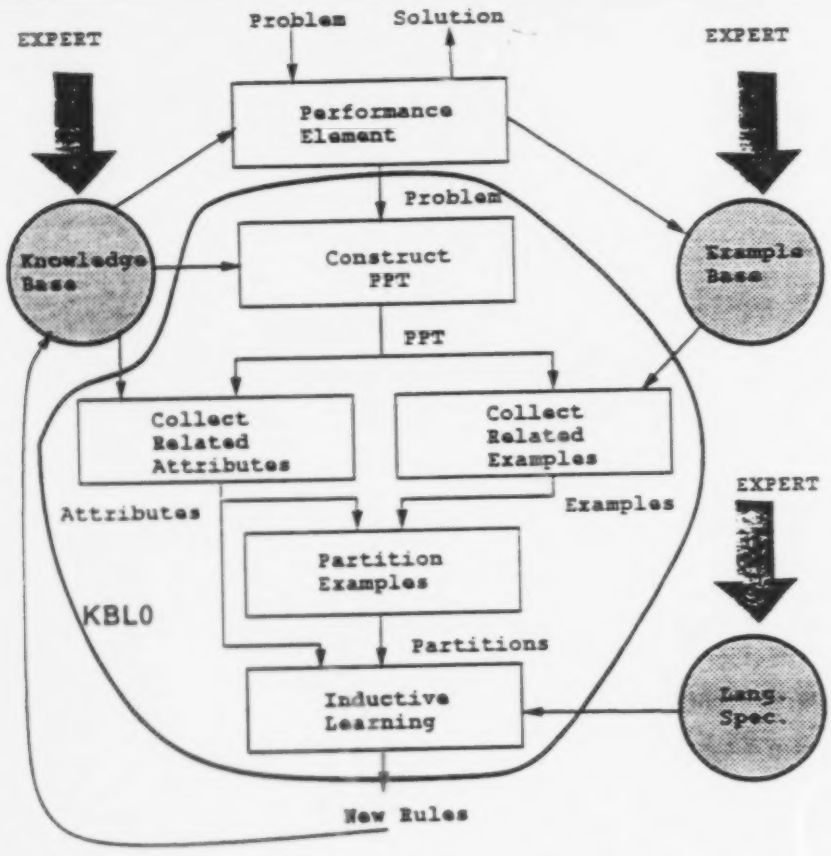


Figure 4.1: KBLØ System Overview

discussion of KBEG, an algorithm for discovering equations used to partition a set of examples (Section 4.2). In Section 4.3, empirical results are used to compare KBLØ and ID3 [Quinlan, 1986] on a simple domain.

4.1 The KBLØ System

A knowledge-based learning system extends an incomplete knowledge base by adding rules that solve a currently unsolvable problem. Using the domain knowledge, the system constructs a partial-proof tree to explain portions of the solution for the problem. The first ungrounded node in the tree indicates where knowledge is missing in the knowledge base. The grounded nodes of the tree are used to determine a context for the missing rule. The system uses the

context to collect solved examples and then performs induction over the examples to infer the missing knowledge. Figure 4.1 illustrates the interactions of the major components of KBLØ. This diagram replaces the KBL box of Figure 3.7 with details of the system. Knowledge in a KBL system includes *examples*, and a *language specification* in addition to domain knowledge rules.

4.1.1 Representation

KBLØ combines two types of knowledge and a language specification to guide rule induction. Each form of information has its own characteristics. To help explain the KBLØ system, knowledge and examples from the PHAST system will be used. The PHasing And Signal Timing system (PHAST) [Linkenheld *et al.*, 1989] is a knowledge-based expert system that designs the phase plan, cycle length, phase sequence, and green time allocation for an isolated traffic intersection. The design strategy of PHAST is similar to that of expert traffic engineers: generate fairly good initial phase and timing plans, evaluate the performance of the signalized intersection, and then modify the plans to address inadequate performance. The input to the PHAST system is the traffic volume, number of lanes in each direction, and characteristics of each lane. From the intersection description, the system generates the phases[2] and the green times for each phase.

4.1.1.1 Examples

Examples represent knowledge of previously solved problems. An example consists of two parts, the initial problem given to the expert system (problem facts) and the solution for that data. The example's problem component is a conjunction of literals consisting of primitive predicates and arguments describing the intersection. The primitive predicates in the PHAST system are approach-type, lane-type, lane-thru-volume, lane-left-turn-volume, etc. The solution of the problem is provided as a grounded instance of the top-level goal. For the PHAST system, `DESIGN-SIGNAL` is the top level goal. The predicate has over 20 arguments that are defined by the system. The arguments indicate which phases are active, phase volume, cycle times, cycle length, and the green times for each phase.

[2] A phase is one state of green lights. For example, a dual left phase occurs when left turn lanes from opposing approaches have left turn arrows. A through (thru) phase is when lanes of opposing approaches have green lights.

4.1.1.2 Domain Rules

Knowledge about solving problems is expressed in domain rules that decompose the problem into subproblems. In KBLØ, rules are represented as Horn clauses. The deductive engine for KBLØ allows an additional cut operator, /, for segmenting independent subproblems. By separating independent parts of the problem, the system is able to determine where the knowledge is missing from the knowledge base. The rules given to the KBLØ system are of the form

```
(Consequent <- Antecedents)
```

For example, a rule from the PHAST system for computing the volume of the opposing left lane is

```
((compute-opposing-left ?x ?v) <-
    (approach-opposing-approach ?x ?op)
    (lane-approach ?lane ?op)
    (left-lane-type ?lane)
    (lane-left-turn-volume ?lane ?v))
```

4.1.1.3 Language Specification

The language specification is used to define the acceptable values for the different literals of the domain knowledge. To declare that the predicate `lane-type` takes two arguments, of which the first must be a lane and the second a specific lane type, one provides KBLØ with

```
(lang-spec lane-type
   (feature :type lane)
   (value :nominal
              (left left-thru
               left-thru-right
               thru thru-right right)))
```

The function `lang-spec` processes the specifications for predicates. It builds accessor functions to the parts of the sentences and stores the knowledge about the language forms so that the modules performing induction can understand the language being used.

The format of the `lang-spec` statement is a name followed by the definitions that define the arguments of the predicate. The names for the arguments in the above example are feature and value. The created accessor functions are *lane-feature* and *lane-value*. Each argument has the form: `(name kind values)`, where name is optional but if given used to build the accessor. Kind is one of `:nominal`, `:linear`, `:real`, or `:type`. For `:nominal` and `:linear` a single value is supplied that lists the acceptable values. In the case of `:linear`, the values are listed from smallest to largest. The kind `:real` is provided with lower and upper bounds; `-inf` and `+inf` are accepted to represent negative and positive infinity respectively. `:type` is special in that the single value provided names an object type to which this value must be bound. The provided value is used to look up a function that insures the variable is bound to the proper type of object during program execution.

4.1.2 System Components

There are many components to a knowledge-based learning system like KBLØ. Interactions between modules allow the system to use as much knowledge as possible in directing the induction of new rules. Figure 4.1 depicts these major activities of the system:

1. Construct the partial-proof tree.
2. Collect examples based upon the partial-proof tree.
3. Retrieve attributes related to the current goal.
4. Partition the examples into categories useful for learning.
5. Induce new rules.

The rest of this section discusses these steps in more detail.

4.1.2.1 Constructing Partial-Proof Trees

A backward-chaining deductive retrieval system is used to build the partial-proof trees (PPT). The system is given a goal to prove and using the domain knowledge an explanation is constructed. If a complete proof tree can be built, the performance element would not have sent the problem to KBLØ. If a complete explanation cannot be constructed for the problem,

the system must determine what knowledge is missing. KBLØ uses a special cut operator to help identify what concept needs to be learned.

The deductive retrieval system has two cut operators. The standard cut operator "!" is used to make processing in the system more efficient (as done in PROLOG programs). The special cut operator "/" segments the rules of the knowledge base into independent sections. When a problem fails to have an explanation constructed, the system determines what needs to be learned by backtracking to the closest enclosing / cut. Rules with the / cut provide more knowledge to the learning system than rules without it. Assigning blame for the failed explanation would be difficult without the special cut operator because any goal in the knowledge base could potentially be extended by learning a new rule. A system could always learn additional rules for the top level goal; this is what standard empirical learning systems do.

4.1.2.2 Collecting Examples

An important aspect of the KBL approach is retrieving relevant examples from a set of solved examples. The system uses the partial-proof tree to construct a context for the unsolved goal. The context consists of the ungrounded version of the PPT leaves. These literals are a set of constraints applied to each of the solved examples. When an example unifies with the context, it is used during the induction of a new rule. As an example, Figure 4.2 contains a simple partial-proof tree. The node that could not be derived is `(get-opp-left-vol ?ap ?vol2)` (marked with ???). The context for the problem is `(left-turn-vol ?l ?vol1)` ∧ `(< 0 ?vol1)` ∧ `(lane-approach ?l ?a1)` ∧ `(approach-opp-appr ?a1 ?ap)`. The node

```
(= ?ltd (- ?vol1 ?vol2))
```

is not used in the context because it is not grounded.

The system collects relevant examples for the learning process rather than using all the examples or having the user select the examples. KBLØ assumes that examples matching the context of the failed problem are closely related to the problem and can be used for learning. The examples' solutions are not necessarily derivable from the current domain knowledge. The system can learn from examples that are not currently solvable with the domain knowledge. As mentioned previously, the solutions to the examples may come from the user.

Due to the backtracking mechanism used to find the most recent enclosing / cut, the produced partial-proof trees are grounded on the left side, but ungrounded on the right (as de-

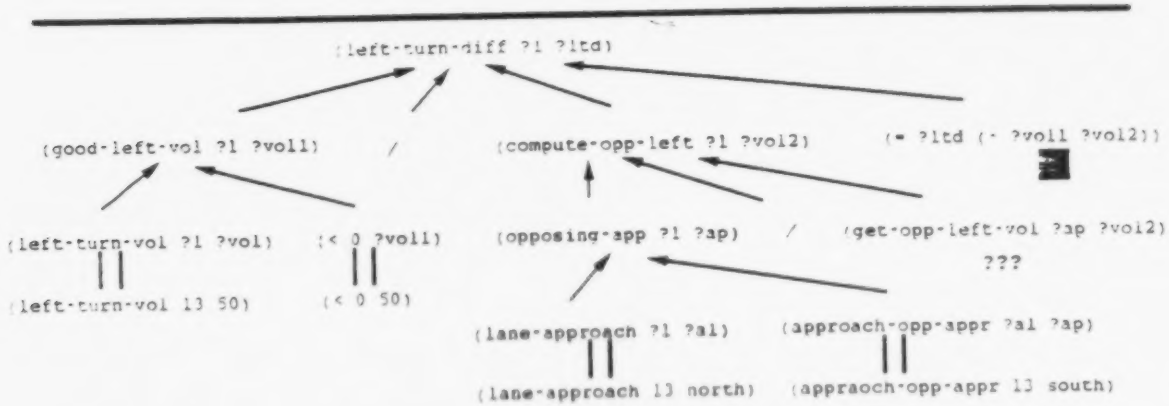


Figure 4.2: Simple Partial-Proof Tree

termined by the form of the rule, which is usually determined by binding constraints). The grounded nodes of the PPT are used to find relevant examples. Thus, the concepts that depend upon the value of a failing literal do not play a role in selecting the examples.

4.1.2.3 Retrieving Related Attributes

Problems from engineering and other real world domains have many attributes. The smallest problem for the PHAST system has 110 attributes. Although empirical learning systems such as ID3 [Quinlan, 1986] can handle large numbers of attributes, the chance of finding a spurious result increases as more irrelevant attributes are used for each example. There are two methods to reduce the chance of an empirical learning system deriving a spurious relationship:

1. Provide the system with a set of examples that densely covers the problem space.

2. Reduce the number of attributes for each example.

The first approach is often not feasible for engineering domains. Even if a program generates examples there are too many combinations of attribute values. Reducing the number of attributes is possible in a KBL system. The user must perform this task himself for traditional empirical learning systems. KBLØ finds attributes for the unsolved goal using rules in the knowledge base for the same goal. The attributes referenced in those rules are preferred for the new rules created.

BEST COPY AVAILABLE

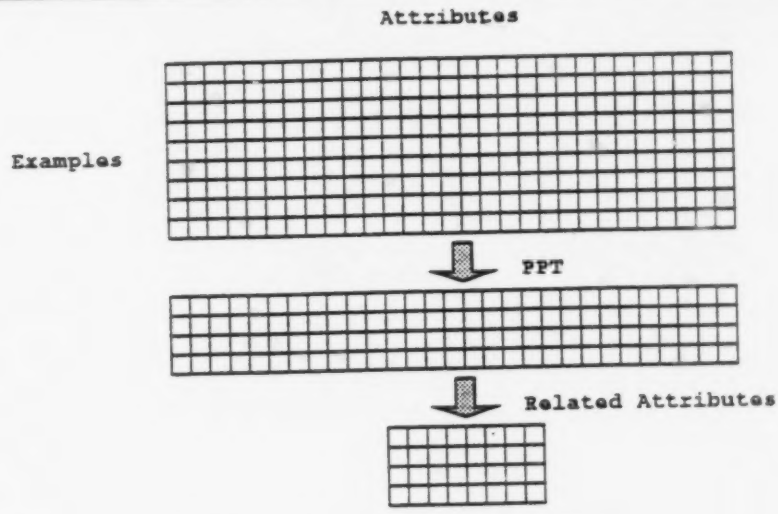


Figure 4.3: Reducing Example Space

The partial-proof tree and the related attributes work in conjunction to reduce the search space for the induction system. Figure 4.3 illustrates how the partial-proof tree reduces the number of examples used for learning and the related attributes reduces the size of each example.

4.1.2.4 Partitioning Examples Into Categories

It is assumed that all solved examples meeting the constraints of the context determined by the partial-proof tree are relevant to the concept to be learned. The examples may be relevant, but one conjunctive rule does not necessarily cover all of them. In order to find better rules, these examples must be partitioned into classes. By partitioning examples into classes before induction, the system can use events similar in form as near misses, making it easier to induce more specific rules.

The examples are partitioned on the missing value of the unsolved goal. For example, in Figure 4.2 the literal (`get-opp-left-volume ?ap ?vol2`) is the unsolved goal and `vol2` is the variable that has not been bound. Thus the examples will be partitioned based upon the value of this variable in their solution.

When the unbound variable is of type nominal or linear, each observed value is placed in a separate category. At worst this makes the induced rules too specific. When the variable is a real number the partitioning is more complex. For engineering problems numeric values are usually calculated from other known values, so simple numeric clustering techniques rarely produce meaningful results. Instead, the examples are partitioned based upon formulas derived for the events. A set of formulas is developed to relate attributes of examples to the value of the variable missing in the unsolved goal. Each partition is described by a formula. The *knowledge-based equation generator* (KBEG) system was developed to perform this task. The KBEG system is described in Section 4.2.

4.1.2.5 Inducing New Rules

The induction of new rules is not difficult at this point. Any number of empirical learning systems could be used to induce a rule. KBLØ uses a Common LISP version of the INDUCE algorithm [Hoff *et al.*, 1983]. One advantage of using the INDUCE algorithm is that it handles structured attributes. INDUCE is given the values for the relevant attributes that were found earlier in the processing. The system produces rules for each partition of examples presented. While rules are being learned for one partition of examples, the examples of the other partitions take the role of negative events.

When the unknown variable for the goal is a real number, the KBEG formula that partitions the events is used as the class description. When using standard empirical learning systems, the user classifies events as positive or negative. In KBLØ, the system has created the classes for the events. The induced rule must have a definition of the class that the user can understand and use. For example, in one run with the PHAST data, KBEG found a partition described by:

```
IF [CYCLE-LENGTH = 51] THEN
   46920 = (* C-CVOL CYCLE-LENGTH)
```

In words, if the cycle length of an event is 51, the event belongs in the partition where cycle length times cycle volume (`c-cvol`) is 46,920. For illustrative purposes, let the set of examples in this partition be called `c-len-51`. This data set was described by INDUCE as:

```
IF [(c-cvol =  790)] THEN c-len-51
```

INDUCE found a description of the data set based upon the cycle volume. KBLØ combines these rules to get:

```
IF [(c-cvol = 790)] THEN
    CYCLE-LENGTH is determined by
    46920 = (* C-CVOL CYCLE-LENGTH)
```

This rule can be used to compute the cycle length when the cycle volume for an example equals 790.

4.1.3 KBLØ Summary

This section has described the KBLØ system, an implemented knowledge-based learning system. The system is invoked when the expert system cannot solve a problem. A partial-proof tree is constructed to identify where knowledge is missing in the domain theory. The PPT is used to collect examples from a database of solved examples. Attributes related to the unsolved goal are found and used to limit the number of attributes the induction system must consider. The examples are partitioned into meaningful categories to facilitate learning. KBL systems work within the structure of an existing knowledge base to induce new rule knowledge from information in examples. Empirical learners must either try to work with all the examples and attributes or require the user to select the appropriate examples and the relevant attributes.

Figure 4.4 demonstrates how knowledge is used by the KBLØ system. Domain knowledge is used to create the partial-proof tree (PPT) for an unsolvable goal. The goal is unsolvable because the domain knowledge is incomplete. The partial-proof tree is used to find relevant examples from the example library to be used during induction. The examples that match the proof structure are selected for use in induction. Using the examples selected, the induction module is able to find a new rule for the concept. The new rule completes the partial-proof tree. KBLØ has been tested with a knowledge base for designing phase signals for traffic intersection lights.

4.2 Knowledge-Based Equation Generator

Partitioning examples for induction is important in preparing examples for induction (Figure 4.1). The knowledge-based equation generator (KBEG) system was developed to meet the

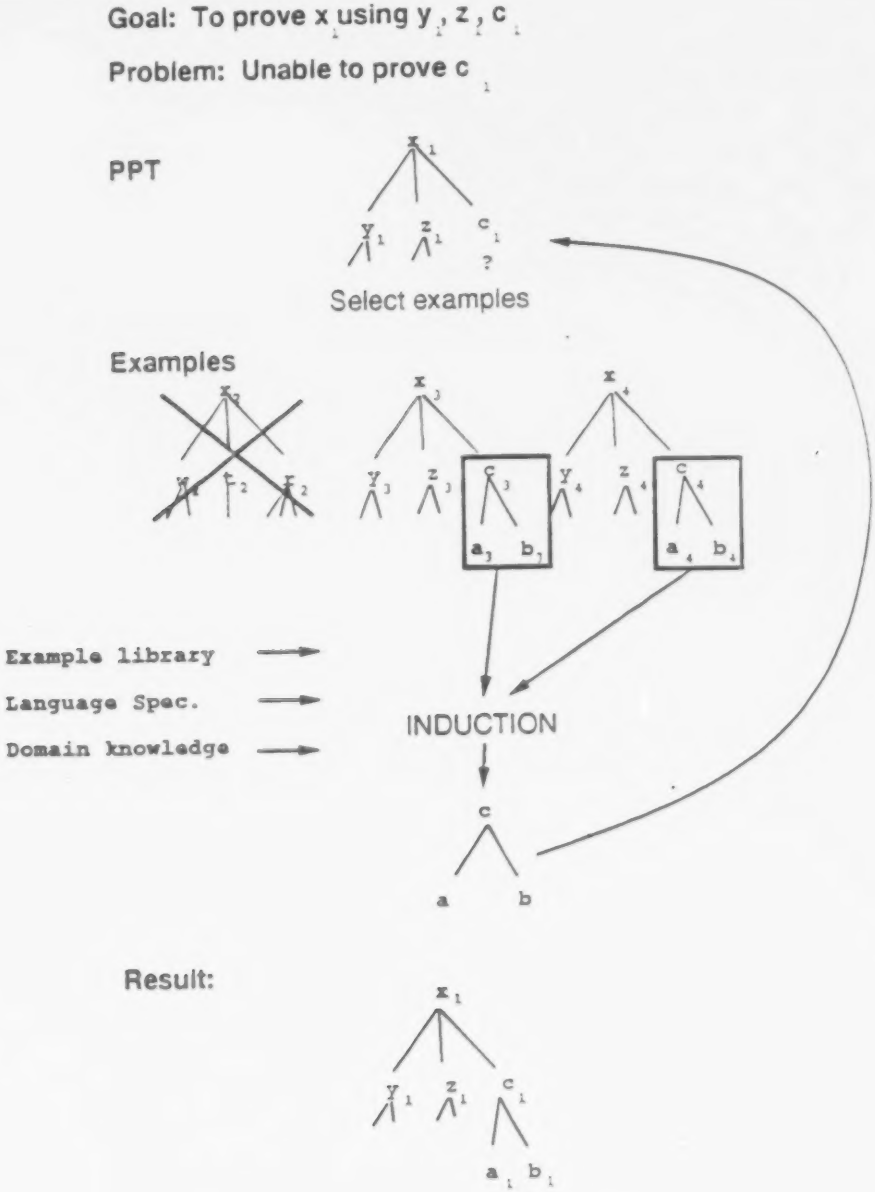


Figure 4.4: Knowledge Flow of KBLØ

partitioning needs of KBLØ. KBEG is also useful as a stand alone system because applying divide and conquer strategies is an effective problem-solving method [Whitehall *et al.*, 1990]. Phenomena are divided into categories to facilitate understanding and reduce the cognitive burden of reasoning about many individual instances. The dividing or partitioning of observed events into useful classes facilities human learning. It also helps computers in solving engineering problems. Recent research in applying machine learning techniques to engineering problems has demonstrated the importance of conceptual clustering techniques in engineering domains [Lu and Chen, 1987, Lu, 1989]. Though conceptual clustering programs [Michalski and Stepp, 1983, Stepp, 1984, Stepp and Michalski, 1986b, Pitt and Reinke, 1988] produce clusterings superior to those produced by numerical clustering methods, one aspect of engineering data is not handled adequately by these systems. Mathematical relations existing between attributes of the events are not used to partition events. Scientific discovery systems [Falkenhainer and Michalski, 1986, Langley *et al.*, 1987] are able to find formulas relating attributes, but the operating assumptions of these systems do not facilitate their use as clustering algorithms. To overcome the limitations of current clustering systems and scientific discovery programs, the knowledge-based equation generator system was developed. The next subsection describes numerical and conceptual clustering and motivation for the development of the KBEG system. Subsection 4.2.2 describes the KBEG system. A demonstration of the KBEG system is provided in Subsection 4.2.3. This section closes with some future research in this area.

4.2.1 Motivation

Partitioning a large set of items into a small number of groups allows one to obtain a better perspective on the differences between items in one group and those in another group. At the same time, the commonalities of all the items in a single group can be emphasized. Much of science is involved with the partitioning of natural phenomenon into the appropriate classes. Engineering endeavors can also benefit from partitioning events. Group technology is one area that is directly concerned with this task [Ham, 1982]. When a set of parts can be thought of as "being the same" in some respect, it facilitates a number of tasks related to those parts. For example, parts in the same group should have common functions and they should have similar steps in manufacturing those parts. An engineer could consider other parts in the same class when designing. Having a classification allows the engineer to focus on the common aspects

of parts in a particular group, thus reducing the complexity of dealing with a large number of individual parts at once.

Though there is great benefit to clustering, it is a difficult task. Computer-based approaches have been used to cluster some events with great success. Numerical clustering techniques work with data that has numerical attributes. One approach to numerical taxonomy is to apply a distance metric between every pair of examples. The partitions are formed by grouping items according to a distance metric. The pair of examples that have the minimum distance are merged to form a composite group. This process continues until only one group remains. The taxonomy is represented by the tree of merging operations performed. Numerical taxonomy is efficient (fast) and useful when the distance metric measures the relevant parameters of the problem. There are two limitations of such an approach:

1. The distance metric requires all the attributes to be numbers.

2. The attributes must be scaled so that appropriate weighting is given to all attributes.

A desired benefit of clustering is to allow humans to understand the problem space and provide a vocabulary for discussing sections of the space. Numerical clustering techniques do not generate human-friendly descriptions of the categories to be constructed.

Conceptual clustering overcomes the major problem with numerical clustering by clustering items based on *conceptual cohesiveness*, a measure that takes into account the similarity between objects and the concepts available to describe the clusters. A conceptual clustering system considers not only the distance between events when constructing a clustering, but additionally, the description used for the clustering. These systems generalize event descriptions in order to produce concise descriptions of the events in the partitions. Because conceptual clustering systems manipulate the language of the problem space, they can deal with nominal type attributes as well as numerical attributes.

Both numerical clustering and conceptual clustering are useful for many problems. But there are some problems neither system is designed to handle. In engineering domains, the important relations between examples are not always identified by specific attribute values. Instead, mathematical formulas often describe important relations. For example, the items of one class might be defined as those events where the value of attribute x multiplied by the value for attribute y equals 5. Another class could be defined where the value of the multiplication

	Events						
x	4	20	2	3	6	15	16
y	2	10	1	1	2	5	8
x/y	2	2	2	3	3	3	2

Table 4.1: Unclusterable Data

is a different constant, such as 8. Numerical and conceptual clustering systems cannot find clusterings based upon these types of relationships.

The knowledge-based equation generator system was developed to find common relationships between events with no values in common, but where a mathematical formula can be found to relate attributes of the events. The relationship discovered is the formula used to relate attributes. Table 4.1 presents a simple set of examples. Each input event has values for two attributes, x and y. If a conceptual clustering system were given these events no interesting clusters could be discovered. But if a system computes x/y, two interesting classes are discovered: $x/y = 2$ and $x/y = 3$.

The shortcoming of current conceptual clustering techniques is that mathematical relationships cannot be used to partition the data. Scientific discovery systems like ABACUS [Falkenhainer and Michalski, 1986] can discover such formulas, but it makes assumptions that do not apply when working in engineering domains or when multiple classes are desired. ABACUS assumes the user has no prior knowledge about the relative importance of attributes and that a *single* formula should be used to describe all the data whenever possible. KBEG uses knowledge about which attributes are most important to guide the selection of the attributes used in the formula. This knowledge is available in many engineering domains. Additionally, KBEG does not assume that a single formula can explain the data; it tries to find multiple, simple formulas, each of which covers a small subset of the input events. Table 4.2 presents a data set upon which ABACUS and KBEG were run. ABACUS discovered the following formula to describe the whole set:

$$2(y^2x) + (yx^2) = y^3 + 2x^3$$

KBEG on the other hand produced two formulas:

IF z = red THEN $1/2 = x/y$

IF z = blue THEN $1 = x/y$

	Events							
x	1	2	3	4	5	6	7	8
y	2	4	6	8	10	6	7	8
z	red	red	red	red	red	blue	blue	blue

Table 4.2: Data set used by ABACUS and KBEG

KBEG was developed when ABACUS was unable to produce the results needed by KBLØwhen dealing with engineering problems. The results produced by ABACUS are useful when a single equation should cover all the data. But for partitioning a set of examples into categories, KBEG results are better, because the resulting formulas are less complex and more than one class is found.

4.2.2 KBEG

The knowledge-based equation generator is based upon the ABACUS system. Before presenting the details of KBEG, a brief introduction to ABACUS is provided.

4.2.2.1 ABACUS Overview

ABACUS is a quantitative discovery system. The system is provided a set of examples and from those examples a mathematical formula relating a subset of the variables that produces a constant value is found. Table 4.3 demonstrates how ABACUS works. Each input event has an x and y value. From this information, the system discovers the relation between x and y, namely $x/y = 2$. Basically, ABACUS combines variables using mathematical relations to derive constant values. There are two main components to ABACUS, equation discovery and precondition generation. Equation discovery depends upon the notion of qualitative proportionality to direct the search for expressions that explain the data. If more than one expression is found, the system uses the precondition generation phase to construct a precondition for the application of the equation. A version of AQ [Michalski, 1985] is used for this part of the algorithm.

Equation discovery creates new variables based upon the proportionality among pairs of existing variables. The variable x is *quantitatively proportional* to y if for a percentage of the instances (a system parameter), the values of x rise while the values of y rise (with other

x	y	x/y
4	2	2
20	10	2
2	1	2
16	8	2

Table 4.3: Simple ABACUS Example

attribute values remaining constant). Likewise, x is *inversely quantitatively proportional* to y if the value of x decreases as y increases. A proportionality graph can then be derived, where the nodes are labeled by the variables and the links represent proportionality. The system can then use this graph to determine where to create new variables. If x and y are quantitatively proportional, ABACUS creates a variable equal to the quotient of x and y and a variable equal to the difference of x and y. If x and y are inversely quantitatively proportional ABACUS creates two variables, one equal to the product of x and y, the other set to the sum of x and y.

Unrestricted, the generation of variables quickly moves into an infinite search space. ABACUS uses two special search methods to restrict the generation of variables: proportionality graph search and suspension search. Proportionality graph search is used only at the start of equation discovery. Proportionality graph search uses only the nodes within a cycle to create variables using the rules described above. After this one level of proportionality graph search, the system uses suspension search. Suspension search is a beam search where the beam is determined by the nodes that are allowed to be active.

4.2.2.2 KBEG Overview

KBEG finds the simplest formulas relating selected event attributes to describe a subset of the examples. The system gives preference to formulas containing variables the user declares relevant (by assigning a numeric worth to each attribute). The system also allows the user to provide a tolerance in the computation of the constant for the formula. This side steps the problems of exact match in continuous variables.

Like ABACUS, the system finds formulas based upon proportionality of attribute values, but KBEG uses both *qualitative* proportionality and *quantitative* proportionality. Qualitative proportionality is used to find general relationships between attributes. Once the qualitative

```
Given two attributes (columns):
Compute qualitative proportionality between attribute values
Break data into monotonic groups
   (allow for noise in doing this)
Do for each group
   If the group > minimum required size
      If group is + monotonic
         Compute x − y, y − x, x/y
      Else
         Compute x * y, x + y
      Endif
      Assign a new attribute (column) to the new values
      If the new attribute has constant value for a set of examples
         Return a new expression with examples covered
      Else
         Add attribute to the processing list
         Compute worth of the new variable
      Endif
   Endif
Enddo
```

Figure 4.5: KBEG algorithm kernel

proportionality between two attributes is known, a quantitative value is computed. If there is a positive relation between the attributes x and y, then $x - y$, $y - x$, and x/y are computed. If there is a negative relation between the attributes, xy and $x+y$ are computed. Unlike ABACUS, KBEG uses the qualitative proportionality to determine where to create subsets of the data. Where the proportionality changes between positive and negative, the system segments the events. The quantitative proportionality is used to determine when a formula has been found that covers a subset of the data. Figure 4.5 explains how this part of the algorithm works. This is the kernel of the KBEG system.

The system works with two attributes at once. The qualitative proportionality is computed between the events in the columns. Using the proportionality, the events are divided into two groups, positive and negative proportionality. In order for the system to attempt to find a formula for the group, it must contain at least a specified number of examples. When enough examples are in the group, a number of new attributes (columns) are created. The exact

attributes created depends upon the type of group. After the values for each new attribute have been created for the group, the values are checked. If the column values are constant for the examples, a formula has been found and that formula is returned. If the new column's values are not a constant, then tasks using the new attribute are added to the agenda.

KBEG computes qualitative proportionality differently than ABACUS. In order for ABACUS to find the quantitative proportionality of two attributes, all other attributes of the examples must be held constant. In KBEG, the other attributes do not need to be held constant to find the proportionality. Instead, the events are sorted by the two attributes, the sort key is the concatenation of one attribute with the other giving the effect of sorting one attribute within the context of the other. The system then observes the proportionality of the two attributes. Although the approach used by KBEG to determine qualitative proportionality is more prone to error than the method used by ABACUS, it is more useful for engineering data. The mechanism used by ABACUS is useful when the data is collected from an experiment where all the conditions except two are held constant. With data collected in engineering domains, this consistency in the data is rare.

An agenda mechanism is used to control the large search space of KBEG. Each task of the agenda contains the attributes to be compared and the events to be covered. The tasks are ordered by the worth assigned to the attributes. The formula for the worth of a task is

worth of attribute 1 + worth of attribute 2
− (depth of search * penalty)

The worth assigned to a new attribute is the largest of the two attributes used in its construction. The depth of search is subtracted from the worth of a task to bias the system to find simple formulas. Penalizing formulas that require deep search allows the system to find simple formulas using attributes initially considered irrelevant, instead of finding complex formulas using the attributes believed a priori to be of interest. Depth of search is defined as the maximum number of operators used in constructing a new attribute.

After performing each task (by applying the proportionality algorithm) to create new attributes, new tasks using the new attributes are added to the agenda. If no new attributes were found, the next task is removed from the agenda and executed. A list of previously covered events is maintained and the set of events for the current task intersecting with previously

covered events are removed before starting the task. Thus, the system returns only the first formula found that covers a given event.

A formula is kept only if there is a simple (conjunctive) description of the covered events. The CAQ algorithm [Whitehall *et al.*, 1991] is used to find this description. CAQ is a concept learning system that learns class descriptions (rules for determining class membership) from provided examples for each class. CAQ is a version of the AQ algorithm that handles continuous (real) values in addition to nominal values.

If a conjunctive description cannot be found, the new attribute is used in further processing. A final formula, however, is not created with the new attribute. The system searches for conceptually simple descriptions of the data until all the events are covered by formulas or a specified number of agenda tasks has been performed.

The user provides information about the relative importance for each attribute. This knowledge allows the system to prefer formulas that use those attributes. A number of other parameters are also provided to the system:

1. Maximum number of agendas to be performed.
2. Penalty for search depth.
3. Percentage of total number of events that can remain uncovered.
4. Percentage of events that must be covered for a formula to be considered a concept.
5. Tolerance on constant values allowed (can be either a fixed number or a percentage of the constant found).

These parameters allow the user to control the space covered in the search and the amount of detail required in the formulas.

4.2.3 KBEG Demonstration

In this section, the KBEG system is demonstrated. First, a simple example is presented to show that attribute weighting can allow the system to find interesting results. Then, some results from an engineering problem are shown. These results provide some insight into the usefulness of the KBEG system. No analysis of the quality of the clusters produced by KBEG

x	y	z
1	9	8
2	8	7
3	7	6
4	6	5
5	5	4
6	4	3
7	3	2
8	2	1
9	1	0
10	0	-1

Table 4.4: Data explained by 2 formulas

is provided. It is unclear how such an analysis would be conducted in a domain independent fashion.

4.2.3.1 Effect of Attribute Weight

Table 4 presents data that can be described with two different formulas. The data can be explained by $x + y = 10$ or $x + z = 9$. The ABACUS system can only find the first formula. KBEG can find either formula, depending upon which attribute, y or z, has been given a larger weight. As mentioned previously, this allows the user to help guide the system to provide formulas using the desired attributes.

4.2.3.2 Combustion Engine Data

This section demonstrates the use of KBEG with some data generated by a simulator for finding the ideal fuel-air cycle for combustion engines. KBEG was run 108 times on various amounts of the simulator data (300 examples total) using several attribute weighting schemes. The output of one run is :

```
[9]  IF [243.77182< TI <297.82013]
        [0.7815315< PHI <1.0372715] THEN
        257.63028 = (* PHI TI)
[10] IF [29.002146< ETAGR <36.50244] THEN
        1.2617798 = (+ (- PHI TI) TI)
```

```
[11] IF [0.9991058< PHI <1.1775184] THEN
        0.9030114 = (/ (/ TI PHI) TI)
[2]  REMAINING
```

There are eight attributes for each example in this domain. The attributes are:

- CR — compression ratio
- PI — inlet pressure (BARS)
- TI — inlet temperature (K)
- PHI — fuel/air equivalence ratio
- PE — exhaust pressure (BARS)
- ETANET — net indicated thermal efficiency
- ETAGR — gross indicated thermal efficiency
- ETAVOL — volumetric efficiency

Looking at the results of KBEG above, the number on the left in brackets is the number of events in the class described by the formula. The IF part of the rule defines when the formula is true (as found by CAQ run on the data). For this set of data, there existed two events that were not covered by any of the three formulas, i.e., KBEG was unable to find a formula to cover them. Each uncovered event belongs in its own partition.

4.2.4 Summary of KBEG

One of the major problems with systems that discover equations, is finding only "interesting" equations. There are many equations that a system like KBEG can discover that are uninteresting. For example, the relative magnitude of attributes can effect the equations discovered. If attribute x is a large number and attribute y is a small number, then KBEG can find $(x - y)/x = 1$ because the system allows a tolerance in the constant value. This is clearly an uninteresting equation to find. Future work will focus on solving this problem and finding heuristics to allow KBEG to perform better in engineering domains.

```
(A ?X ?Y ?Z) ← (A1 ?X) / (A2 ?Y) / (A3 ?Z)
(A1 ?X) ← (B1 1) (B2 1) (= ?X 1)
(A1 ?X) ← (= ?X 0)
(A2 ?X) ← (B2 0) (B5 1) (B6 1) (= ?X 1)
(A2 ?X) ← (B2 1) (B7 0) (B6 1) (= ?X 0)
(A2 ?X) ← (B2 1) (B4 0) (= ?X 1)
(A2 ?X) ← (= ?X 0)
```

Figure 4.6: KBLØ Binary Test Domain

In summary, KBEG allows a user to find mathematical relationships between attributes of a data set. The system is biased to prefer formulas using attributes with a greater worth, as provided through the relevancy scores of the attributes. KBEG does not assume a single formula should cover all of the data, thus its results can be used to partition examples to aid an engineer or learning algorithm to perform better.

4.3 Comparing ID3 and KBLØ

This section presents some empirical results that compare the accuracy and speed of learning for ID3[3] and KBLØ. A simple domain was constructed for these experiments, as shown in Figure 4.6. The problems consist of 10 binary-valued attributes. The top most goal, `A`, has three binary-valued argument, where the first variable is determined by the rules for `A1` from Figure 4.6, the second variable by the rules for `A2`. The third variable's value (`A3`) is determine by the rules:

```
(A3 ?X) ← (B2 1) (B6 1) (B9 0) (= ?X 1)
(A3 ?X) ← (B2 0) (B6 1) (B8 1) (= ?X 0)
(A3 ?X) ← (= ?X 0)
```

These rules are not included in the domain theory for KBLØ. The problem space defines 1024 possible values and there are 8 decision classes. This domain has only nonstructured attributes. in order to allow ID3 to process the data.

[3]The version of ID3 used for these tests handles multiple classes.

KBL and ID3 use the training examples in very different ways. KBL uses the examples in the example base (the training examples) only when the performance element fails to solve a problem. Thus, if a problem is never presented to the system requiring knowledge from the examples, that knowledge is never extracted from the examples and put into rule form. In contrast, ID3 looks at all the examples and extracts knowledge from those examples when it is run. These differences complicate the comparison of the systems. For the domain theory given to KBLØ (Figure 4.6), learning will always occur on the first problem presented. Subsequent learning occurs in KBLØ as examples not covered by the extended theory are encountered. The next section describes the testing procedures used to compare ID3 and KBLØ while accounting for the different uses of knowledge.

4.3.1 Testing Method

In each experiment, the learning system was given a set of training examples randomly selected from the problem space. ID3 constructed a decision tree with these training examples, and KBLØ stored the examples in the example base. Next, 25 episodes of the procedure (defined next) where performed. An example was randomly chosen from the problem space (without replacement) and tested on each system. If ID3's decision tree correctly classified the event, the decision tree was unchanged; otherwise, a new decision tree was constructed using the initially provided training set and all examples seen thus far. If the current domain theory of the KBLØ system could not assign a decision value for the event, learning occurred to extend the knowledge base. Each presented example was added to the example base. When either system learned, examples representing all the points from the problem space were tested against the new decision tree or domain theory and the accuracy was recorded. This means the accuracy reported is the absolute accuracy (number of positive examples classified correctly divided by the size of the problem space) of the state of the knowledge (decision tree or domain theory) after the learning occurred. The number of examples in the initial training set was varied in the experiments presented below. For each training set size, 20 runs were performed. The results presented are averaged over the 20 runs for each set of experiments.

By testing each system in an incremental manner, the systems could be compared. In a batch mode, KBL systems do not learn. They must be presented with unsolvable problems to extract the useful knowledge from the examples. This method allows the comparison of

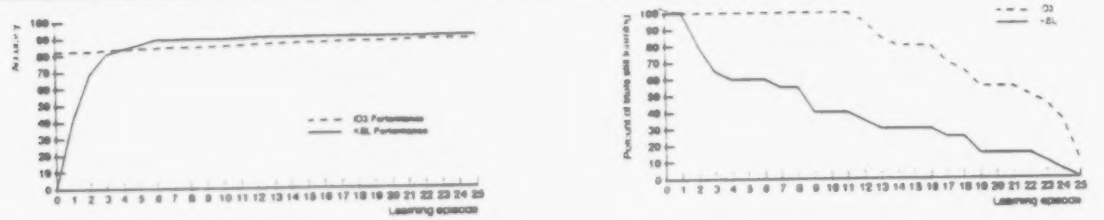


Figure 4.7: System given 50 initial cases

accuracy on the domain as well as a simple comparison of the rate of learning. All the results are presented with two graphs, as done in Figure 4.7. The left graph presents the accuracy of the system as events are seen by the system. The right graph describes how quickly the system learns. This graph indicates what percentage of the runs have not achieved their maximum accuracy at the specified learning episode (the percentage of trials still improving after seeing the examples). For example, at episode 5, 60% of the 20 runs for the KBLØ system have not reached their best results, while 100% of the ID3 runs have not reached their maximum performance. The graph on the right is only useful in the context of the graph on the left. A system might be able to learn quickly, but have low accuracy; the two must be considered together. When two systems tend to have closely related performance, the right hand graph indicates trends showing which system is approaching its peak performance more rapidly.

As an example of the rules learned by KBLØ, the following rules were added to the domain theory on the first episode of the first run when the initial training set had 50 examples:

```
(A3 ?X) ← (B4 0) (B2 1) (B1 1) (B9 1) (= ?X 0)
(A3 ?X) ← (B4 0) (B2 1) (B1 1) (B9 0) (= ?X 1)
```

KBLØ used 9 examples from the example base during the induction of these rules.

4.3.2 Learning Results

In this section, the results of using the procedure described above with initial training sets of 50, 25, 35, 100 and 200 examples are shown. The results of these experiments are presented in Figures 4.7 through 4.11. The graphs indicate there is a relationship between the size of the initial example base and the system performance. Figures 4.10 and 4.11 show the learning

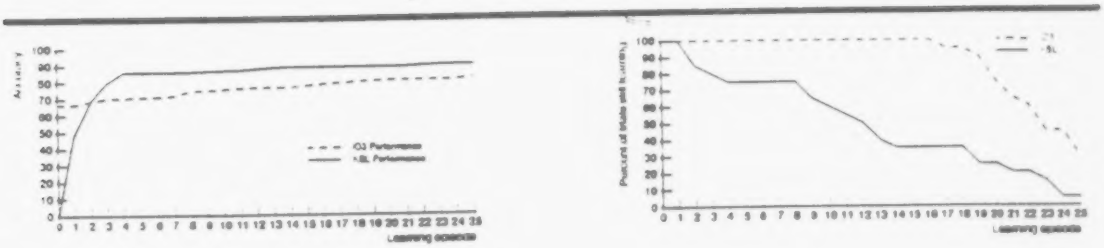


Figure 4.8: System given 25 initial cases

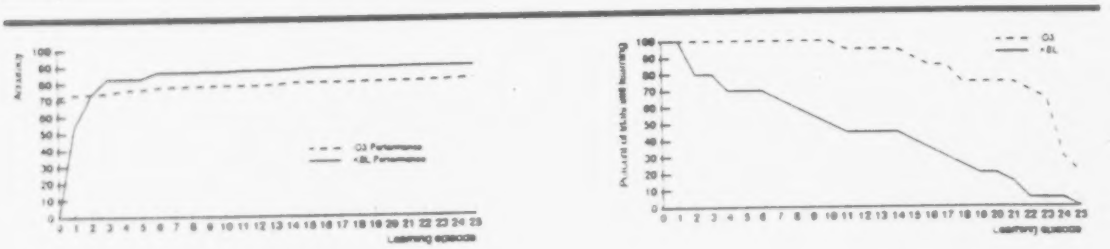


Figure 4.9: System given 35 initial cases

when the training set contains 100 and 200 examples, respectively. In these situations ID3 has slightly higher accuracy than the KBL system. This is to be expected because the number of examples is a significant portion of a problem space containing only 1024 examples. As fewer examples are initially provided to the systems, KBLØ performs better than ID3. Figure 4.7 presents the data when the initial data set contains 50 examples. Not only does KBLØ have higher accuracy, it also obtains that accuracy after seeing just a few examples. The trend continues as fewer examples are initially provided (Figures 4.8 and 4.9).

4.3.3 Results on Stability

KBL systems are more stable in the accuracy of the produced rule set than ID3. This stability is a result of combining the two different forms of knowledge so that one form of knowledge compensates for the inadequacies of the other form. The examples can easily provide knowledge not yet encoded in rule form and demonstrate exceptions to existing rules. The rules can indicate which attributes are important, even if there are not enough examples to bare this out.

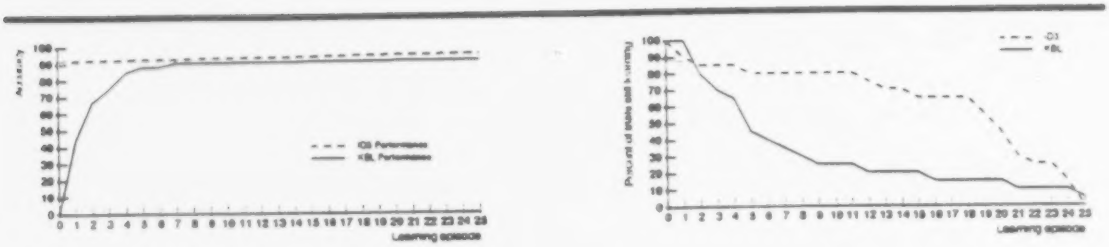


Figure 4.10: System given 100 initial cases

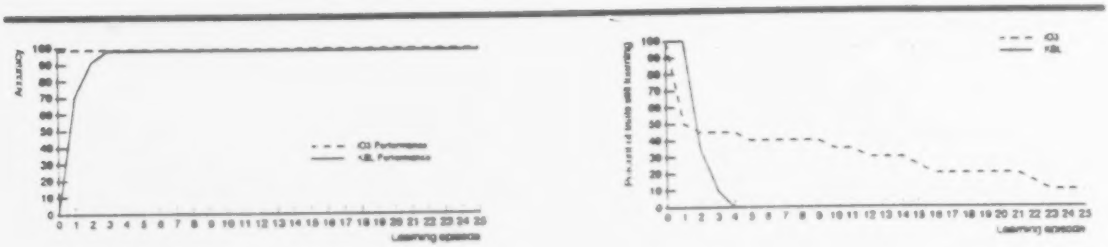


Figure 4.11: System given 200 initial cases

These experiments demonstrate the stabilizing effects domain knowledge has in the induction process. As the number of examples decreases, the domain knowledge plays a more dominate role and helps the system to maintain its performance. One measure of stability is the difference between the accuracy of the systems when presented with few initial examples and many initial examples. Table 4.5 compares the accuracy of ID3 and KBLØ after seeing 25 examples in addition to the initial data set. The difference between the accuracy of ID3 when given 25 and 200 initial examples is over twice that of KBLØ. Table 4.6 presents the same information when the systems have seen only 10 examples since the initial training set. KBLØ is more stable because the domain knowledge can direct the induction of rules even when there are fewer examples to use during induction.

4.4 Conclusion

Explanation-based learning and empirical learning techniques place limiting constraints on the knowledge to be used for learning. Principally, they can use only one type of knowledge.

System	Number of initial examples		
	25	200	Δ
ID3	82.1	99.3	17.2
KBLØ	89.6	98.2	8.6

Table 4.5: Percent Accuracy at Episode 25

System	Number of initial examples		
	25	200	Δ
ID3	74.2	98.8	24.6
KBLØ	86.6	98.2	11.6

Table 4.6: Percent Accuracy at Episode 10

Knowledge-based learning is an alternative approach that allows some knowledge in rule form and some in the form of examples. KBLØ demonstrates that allowing domain knowledge to help guide induction is not only feasible, but allows the system to learn with fewer examples than required by pure empirical techniques. As a result of injecting rule knowledge into the induction process, the results of the learning system do not vary as greatly as when that knowledge is missing.

There are negative aspects of using both forms of knowledge as well. KBLØ does not take a global view of the data like ID3. Therefore, when a large number of examples are present, KBLØ is slower to learn the concept because only subsets of the examples used at one time. KBLØ does not easily handle the case where an example contradicts a rule in the knowledge base. KBLØ assumes the knowledge base is correct; this assumption is relaxed in the KBL1 system which is presented in the next chapter.

Chapter 5

KBL1

In general, determining where a knowledge base needs to be repaired or extended is difficult. This was avoided in the KBLØ system by requiring the person constructing the initial knowledge base to place the cut operator, /, before subgoals that can have rules learned for them. In this chapter the problem of determining repair location is described in more detail along with a solution. *Rule-Structured Induction* (RSI) is an inductive approach to determining where knowledge is missing from a domain theory. KBL1 uses rule-structured induction to learn rules more focused by the problem. RSI also frees the knowledge engineer from determining the locations most suitable for learning in the existing rule structure. Using RSI permits the KBL1 system to address the problem of obtaining the value of the *decision variable* for an unsolved subgoal. The decision variable is the variable that is assigned a value by solving (proving) the subgoal. The value for this variable is used to partition the collected examples, as in KBLØ. In the KBLØ system it was assumed that a solution for the subgoal was provided. When the knowledge engineer indicates where learning can occur via the special cut operator the subgoals allowed for induced rule consequences are identified. This makes it easy to write procedures for converting the provided top-level solution of a problem into an instantiated subgoal. For KBL1, the learning can occur at any subgoal. Thus, the learning system must be capable of determining the decision variable's values automatically.

KBL1 addresses the problem of where is the knowledge from the example base to be added to the rules? One aspect of this problem is inherent in many analytical learning systems, mainly, determining the correct proof construct to use in the learning (generalization) phase.

Many explanation-based learning systems use the first proof (tree) constructed for the goal. While this is a practical approach, the first proof tree might not be the only (and therefore not necessarily the best) proof to use. In knowledge-based learning, the problem is more severe. Analytical learning systems know the proof solves the problem, so even if the best proof is not generalized, the resulting rule will still be correct. When KBL systems need to learn, any subgoal could be used as the consequent of a new rule. Because there is nothing to indicate exactly which goal (subgoal) has the missing rule, the system could induce a rule for a goal with the correct rules and, thus, over generalize the rules for that goal. To help avoid this danger, rule-structured induction is guided by the following heuristics:

1. Take advantage of the knowledge in the problem by using the (derived) context.

2. Minimize the amount of change (generalization) applied to the knowledge base.

3. Take advantage of the knowledge in existing rules and examples.

The next section provides a more detailed look at the problem and describes the RSI approach to the problem. The second section describes how the values of decision variables for examples can be obtained to aid in rule induction (when that is required). In the third section the system used to partition the events, KBEG-R, is presented. The fourth section describes the induction module for the system and Section 5 describes how to specialize overly general rules. The sixth section describes how search through an AND/OR tree, as would be required for RSI, can be reformulated into a search through an OR tree, allowing heuristic search techniques to be used. Section 7 describes the KBL1 system which is composed of the modules described by the previous sections.

5.1 Rule-Structured Induction

Unlike learning systems that are told *what* to learn, KBL systems must determine where knowledge is missing. Similarity-based learning systems are informed what to learn by the class membership of the provided examples. Explanation-based learning systems are provided with the goal concept; the goal concept is the consequent of the rule to be learned. KBL systems must determine the correct goal for the consequent of the learned rule, which subgoal is missing a rule. Since there is no a priori information about which knowledge is missing from

the knowledge base, inductive inference must be used to identify what is (conjectured to be) the consequent of the new rule.

One approach to identifying missing knowledge is to assume that any unsatisfied subgoal is caused by a missing rule. This is an erroneous approach because backtracking might have been trying to solve the original problem by trying different variable bindings or by using a different rule for a subgoal farther up in the proof structure, thus, avoiding the unsolved subgoal. Yet, if a complete proof cannot be constructed, knowledge must be missing for (at least) one of the unsolved subgoals. If the system backtracks until no partial proof is identified, then the induction mechanism will learn rules for the top-level goal and the system will not take advantage of the knowledge in the system.

To overcome the problem described above, a heuristic approach will be used to identify where the knowledge base needs to be generalized. The goal of RSI is to generalize the knowledge base as little as possible while solving the current problem. The system has a bias against inducing a new rule when an existing rule can be slightly modified to solve the problem. The rules selected for generalization have the most subgoals satisfied and require small inductive leaps to allow the unsatisfied subgoals to be proved for the problem.

5.1.1 Problem Definition

Inducing rules within an existing structure is similar to the problem confronting an incremental learning system. The system is provided an event not covered by the current set of rules. The goal is to extend the set of rules to cover the new event. The version spaces method [Mitchell, 1982] does this for events in an attribute space in which only conjunctive descriptions are allowed. The problem is more complex if events have relational predicates, disjunctive concepts are allowed, or the rules compose a hierarchical structure.

Before defining the RSI approach, a hierarchical rule structure must be defined.

Definition 1 *A* hierarchical rule structure, *R, is a set of rules in which the consequence of one rule is used in the antecedent of another rule. Rule chaining can occur to any finite depth.*

A hierarchical rule set is denoted by

$$
\begin{aligned}
R_0 &\leftarrow C_1 \wedge R_1 \wedge R_2 \wedge \ldots \wedge R_n \\
R_0 &\leftarrow C_2 \wedge R_{n+1} \wedge R_{n+2} \wedge \ldots \wedge R_m \\
&\vdots \\
R_1 &\leftarrow C_3 \wedge R_m \wedge R_{m+1} \wedge \ldots \wedge R_p \\
&\vdots
\end{aligned}
$$

where each C_i is a conjunction of literals

$$c_{i1} \wedge c_{i2} \wedge \ldots \wedge c_{iq}$$

C_i is the primitive portion of the rule, that is, all the c_{ij} literals have predicates that do not occur as the consequent of any rule. R_i is a literal whose predicate is the consequent of at least one rule. R_0 is the highest level goal of the system. The R_i's of a rule set will be called the *chaining* literals of the rule because they can be grounded by chaining through a number of rules. The R_i's for a single consequent do not have to be unique. For clarity, only Horn rules are used in this chapter. A PROLOG program is an example of a hierarchical rule structure.

For generalizing within a rule structure, there is an event e_p which the user has indicated belongs to the class of events defined by a hierarchical rule structure, R. However, the set of rules, R, does not cover the event. The rules in R must be generalized to cover e_p.

As shown in Figure 5.1, there are four general methods to modify a set of rules to cover a new event. (*To cover* means extending the knowledge base so that rules will fire for the facts of the unsolved problem.) The diagram shows a case of a single chaining literal referenced on the left-hand side of the concept rule. The concept rule is

$$\textit{Positiveclass} \leftarrow C \wedge R$$

The four possible solutions are:

1. Generalize both the selected concept rule constants in C and the chaining rule R to cover the new event e_p.
2. Generalize C to cover e_p and drop R from the concept rule.
3. Generalize R to cover e_p and drop C from the concept rule.
4. Create a new rule for the concept, $\textit{Positiveclass} \leftarrow e_p$, thus introducing a new disjunct for the concept described by the positive class.

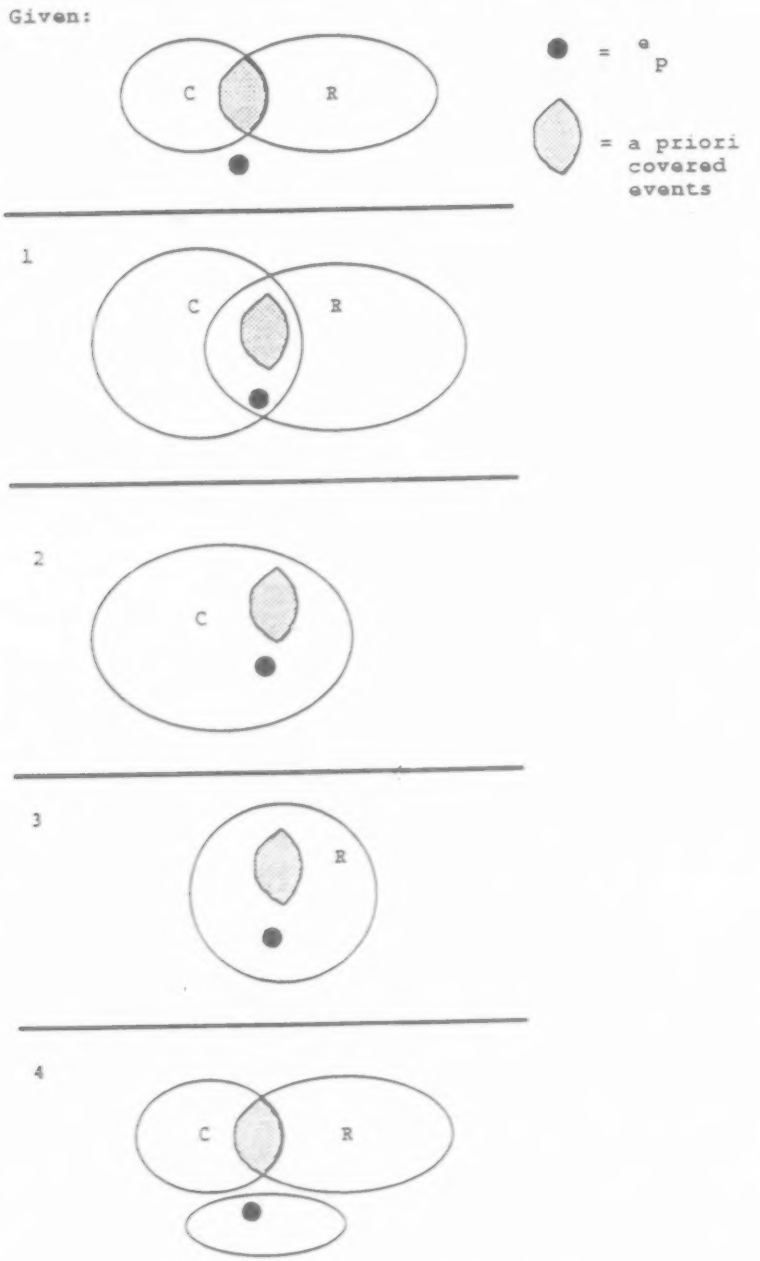


Figure 5.1: Rule-Structured Induction

These are the only operations than can be performed to generalize the coverage of a rule. Note that initially C or R might already cover e_p (Figure 5.1 shows the case where neither initially covers e_p). The combinatorics become more complex when the concept rule has more chaining rules in its antecedent. If the concept is defined by a number of rules, the learning system must choose among these options for each of the rules and determine the best generalization overall. When a number of rules for a concept exist, the system could remove the current rules and perform a batch inductive learning technique to learn a new set of rules. For the system to do this, all the events currently covered by the rules must be available. If a system chooses this option, the chaining rules could be used to infer higher-level (constructed) features for the events.

5.1.2 Assigning Heuristic Goodness Values

To perform rule-structured induction, a numeric value is assigned to each rule indicating the amount of generalization required for it to solve the current problem. Four factors are used in computing the numeric value.

1. **Number of subgoals that exactly match the current problem.** If many antecedents of a rule are grounded with the attributes of the current problem, less generalization is required to make the rule applicable for the problem.

2. **Number of subgoal predicates matched with unmatched arguments.** In this situation, the attributes (predicates) are present in the problem, but their values do not match the values in the antecedents. For example, the antecedent is (`foo 5`) but the problem description has (`foo 3`). Generalization is required if these subgoals are to be used to solve this problem. Guided by the desire to generalize as little as possible, the algorithm prefers using rules with subgoals almost covering the problem. There are degrees of matching. If the current subgoal requires a value for attribute X between 1 and 100, while the value of X is 101, a small leap of faith is required to extend the range for the subgoal. If instead the value for X is 1000, then one should be hesitant to extend the range for this attribute and should consider looking at another rule. The approach should be sensitive to the scale of the measurement.

3. **Number of unmatched subgoal predicates.** An unmatched predicate could either be a chaining predicate or primitive predicate. When a primitive predicate does not match any of the predicates in the current problem, the only generalization technique permitting the rule to fire is to drop the rule antecedent. When a chaining predicate does not match the current problem, none of the rules that have the subgoal as the head can be used to prove the subgoal. In this situation there are two options: create a new rule for the goal, or generalize a rule in the structure for this goal. RSI can be applied to the subgoal to determine if one of the deeper-level rules needs to be modified. The system looks deeper into the structure for the appropriate generalization. If no lower-level rule is modified, the system induces a new rule for the subgoal.

4. **Prefer to generalize longer rules.** Rules with many subgoals are more specific than rules with a few. To minimize the impact of the generalization, the learning system should work with rules unlikely to apply in situations other than the current problem and closely related problems.

In addition to the general guidelines provided above, the user preferences for where the generalizations occurs needs to be considered. The user might prefer adding new rules to modifying existing ones. For example, suppose the existing knowledge base has worked for a large class of problems that make up the bulk of the load for this system. When seeing a new problem, one would be hesitant to change the existing, well-tested rules. In some situations, the user might prefer that rules be added or changed at higher levels in the proof tree instead of making generalizations to subgoals lower in the tree. To accommodate these different preferences, the user provides parameter values that determine which rule should be generalized.

RSI requires five parameter values that are used to assign a numeric score to each rule. These scores determine which modifications to the knowledge base meet the preferences of the user. The rule with the largest score is considered the best. The five score values are:

1. **Perfect match.** The antecedent matches the problem.

2. **Slight extend.** The predicate matches but a nominal value range needs to be extended or a real number range needs to be extended slightly (a threshold parameter).

3. **Extreme extend.** The predicate matches but a real number range needs extended by more than the provided threshold for a slight extension.

4. **Chaining failure.** The subgoal is a chaining predicate where the rules for the predicate are not grounded.

5. **Primitive failure.** A primitive predicate does not match any of the attributes of the problem and would have to be dropped for the rule to work for the problem.

Figure 5.2 contains the algorithm used to compute the numerical value (factor) for a rule. The Find-match algorithm called by Factor determines the degree of match from the five provided scores and returns the appropriate overall score. Figure 5.3 presents one reasonable set of score values. The user that established the values in Figure 5.3 has a strong preference to modify existing rules over adding new rules. This preference is expressed by a low score for chaining failures. A chaining failure is the only type of failure that signals RSI to induce new rules.

5.1.3 Search For Generalization

What has been presented so far shows how RSI can target a rule to generalize from a set of rules. The rule with the value (factor or ratio of generalization values to perfect matches for all components) closest to 1 is designated the best rule to generalize. Generalizing this rule to solve the problem requires less inference, as measured by syntactical changes. Generalizing rules in isolation, however, is insufficient. The system must consider how changing a rule at one level effects the performance of other rules in the system. A simple change at one level might cause complex changes to occur at other levels, thus complicating the proof structure or even making it incorrect for other examples. To help avoid this problem, the system searches through the possible changes and determine the best alternative.

The search performed by RSI is depicted in Figure 5.4. This is a search through an AND/OR tree for the path that satisfies the top-level (problem) goal while requiring the fewest number of modifications to the existing knowledge base. At the root of the tree is the goal for the problem. A number of rules can be used to prove the goal, only one of which is required. Each rule consists of a number of antecedents, each of which must be proved for the rule to prove the goal. A simple backward chaining mechanism such as that described in [Winston, 1984] can be used for this search. Section 5.6 describes how this search can be converted into a

```
Factor(R, E_p, Θ)
; R - rule
; E_p - current problem
; Θ - bindings
Sum = 0
Count = 0
Do for each A in Antecedents(R)
    Sum = Sum + Find-match(AΘ, E_p)
    Count = Count + 1
Enddo
Return(Sum / (Count × perfect-match-value))

Find-match(A, E)
; A - antecedent
; E - problem description
If A ∈ E
    V = perfect-match-value
Else If A matches e ∈ E within tolerance
    V = slight-extend-value
Else If Predicate(A) = Predicate(e), e ∈ E
    V = extreme-extend-value
Else If Chaining-pred?(Predicate(A))
    V = chaining-failure-value
Else
    V = primitive-failure-value
Endif
```

Figure 5.2: Factors Computation

Perfect match	10
Slight extend	7
Extreme extend	6
Chaining failure	4
Primitive failure	0

Figure 5.3: Sample User Preference Values for Generalization

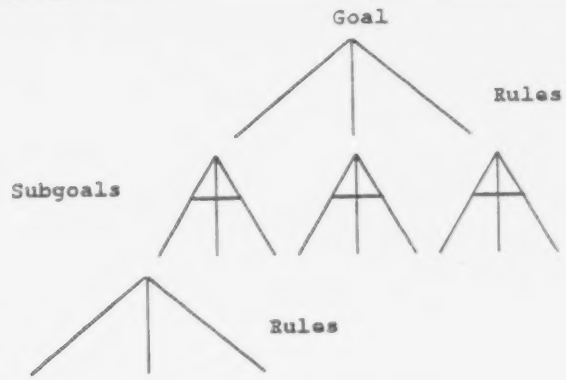


Figure 5.4: RSI Search

search through an OR tree, allowing heuristic search techniques such as branch and bound to be applied.

To summarize RSI, the system searches the rules in the knowledge base to find a proof that is as complete as (heuristically) possible. The system chooses rules to generalize based upon the parameters provided by the user. Those values determine preferences for different methods of extending the knowledge base. When a chaining failure occurs, the system applies learning for the failed subgoal. Although the system is biased against inducing completely new rules when existing rules can be generalized, there are times when that is the best repair for the system. Specifically, if the value for all the rules of a subgoal are below a (parameterized) threshold, say 0.75, then the system will not try to modify any of the existing rules. The factor values in this case indicate that none of the current rules are close enough to solving the current problem to warrant the effort of repairing them. The induction of a new rule is always an alternative for a chaining failure and depending upon the search procedure used, could be performed even when there are other modifications possible to deeper rules. If many changes are involved in repairing the deeper level, it might be advantageous to create a new rule.

5.2 Regression of Variable Values

One of the key aspects of a knowledge-based learning system is the collection of examples for learning. For examples to be useful in learning missing knowledge, the variable values for the subgoals in question must be known. Getting those values is not an easy task. The KBLO

```
(TOP ?x) ← (top-a ?z) (top-b ?y) (top-c ?x ?y ?z)
(top-c ?x ?y ?z) ← (equal ?y blue) (subg2 ?w ?y ?z) (< ?w 5) (= ?x low)
(top-c ?x ?y ?z) ← (equal ?y red) (subg2 ?w ?y ?z) (> ?w 5) (= ?x low)
(top-b ?y) ← (attr-1 ?a) (attr-2 ?b) (SUBG1 ?y ?a ?b)
```

Figure 5.5: Rules from Knowledge Base

system assumed those values were either provided with the examples or procedures were given to construct the grounded subgoals for each example. This was a valid assumption for the KBLØ system, because rules could only be learned for prespecified subgoals. In the more general case the assumption is not valid. Using rule-structured induction, the KBL1 system can potentially learn rules for any subgoal in the knowledge base.

Suppose the system has collected a number of example/solution pairs from the example base that match the partial-proof tree for the problem. (The partial-proof tree is constructed by using the RSI mechanism explained previously.) If the highest-level goal in the system is `(TOP ?x)`, then each example will contain a grounded value for `TOP`, such as `(TOP low)`. The unsolved subgoal for the partial-proof tree would be of the form `(SUBG1 ?x ?y ?z)` where `?y` and `?z` are already bound and the purpose of `SUBG1` is to compute a value for `?x`. For each of the examples collected it is possible to find the appropriate value for `?y` and `?z` by using the partial-proof tree. Unfortunately, there is no easy method for obtaining a value for `?x`. The rule to be induced should compute this value. In order for the KBL method to work, the system needs the value of `?x` for all the (collected) examples. This value is used to partition the examples (if needed) and to construct the missing rule.

It is a nontrivial task to get the value for the unbound variable. A simple example will help clarify this. Figure 5.5 shows some rules from a knowledge base. `TOP` is the root of the proof tree and `SUBG1` is the subgoal (of `top-b`)that cannot be solved for the current problem. The decision variable for `SUBG1` is `?y`. The system needs to compute a value for `?y` for each example from the provided information `(TOP low)`. But as can be seen from the rules, `TOP` has been assigned the value of `low` because of the values of `?y`, `?z`, and `?w`. The value of `?w` also dependents on the value returned by `SUBG1`.

The system cannot always regress values through the rules to get the exact missing value for the decision variable because:

- The decision variable might not ever be used. The subgoal could be a test in the rule to insure certain conditions hold such that the returned value is not important. If the value is not used anywhere else, then regression cannot get the value.
- The rules using the decision variable might not be specific enough to assign a unique value to the variable. It is possible that only a range of values can be defined for the variable.

When a regression approach is used, the best one might get is a range of possible values for the variable. This is problematic because most rule based systems do not allow disjunction in the consequent. One cannot arbitrarily select a value from the range and expect that value to solve the current problem (or others) correctly. For example, using simple regression for the rules in Figure 5.5, the range of values for the result of `SUBG1` is {red, blue}. If the induced rule is constructed using the value blue, the current problem might not be solved because the value for `?w` in the `top-c` rule is greater than 5. The second rule for `top-c` that was used in constructing the range of values cannot be used even though it could work for the given value of `?w` because the value set for `SUBG1` (variable `?y` in the rule) is not red. So, even if the antecedents of the induced rule are correct, the system has produced a rule that does not help solve the problem.

Instead of collecting only the ranges for a variable, the system must find ranges that are applicable in different situations. In other words, a possible worlds approach is taken. The system returns a number of different ranges. But in this case, any value from the range is equivalent to any other value for the computations that follow. In the previous example, red and blue would be in different possible worlds because different rules were used to derive those values.

To formalize this problem the system needs to find a value, V, for the unsolved subgoal, G in the proof tree. The value V must be found within the context of the bindings, Θ, for the current partial-proof tree. Θ is a standard binding list of the form

$$\alpha_1 : \beta_1, \alpha_2 : \beta_2, \ldots, \alpha_n : \beta_n$$

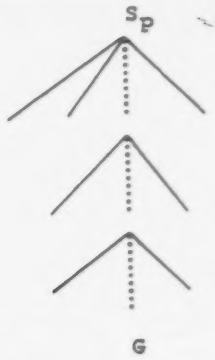


Figure 5.6: Partial-Proof Tree with Highlighted Border

where α_i is a variable and β_i is the binding associated with that variable. Each example in the example base is paired with a solution of the form

$$S = (S_p, S_1, S_2, \ldots, S_k)$$

where S_p is the predicate of the top-level goal and S_i is the value for the ith place of the predicate that should be produced by a correct and complete knowledge base. The regression system works from S to compute a value V.

The knowledge base consists of Horn rules, an example of a rule for the top goal is:

$$(S_p\ ?x_1\ ?x_2\ \ldots ?x_j) \leftarrow (P_1\ ?x_{a1}\ \ldots ?x_{b1})(P_2\ ?x_{a2}\ \ldots ?x_{b2})\ldots(P_n\ ?x_{an}\ \ldots ?x_{bn})$$

From the partial-proof tree we can determine which rule is used to solve the top-level goal, what the proof structure is to the left of G in the PPT, and which rules are used on the *border* of the proof tree. Figure 5.6 presents an example of a partial-proof tree where the border is represented by a dashed line. The bindings made for the variables to the left of the border is Θ. Φ is a binding list that will be constructed for the expressions to the right of the border. The system does not know what the correct structure is to the right of the border, so many possible structures could be constructed. This results in the many possible bindings, and thus the possible worlds.

The binding list Φ is more complex than Θ in that a variable is not bound to a particular expression. Each entry in Φ consists of three parts: (α, ρ, ν). α is the variable from the expression. ρ is the range of values that are known to be acceptable for α. The range is represented as a list of values, e.g., (2 5) indicates $2 \leq \alpha \leq 5$. If the range has been narrowed

to a single value, only that value is maintained. ν binds the value of α to other variables. This is done because when regression is performed, the constraints might not be fixed (single values) as they are in the normal chaining mechanism. As an example, constraining $\alpha_1 = \alpha_2$ might not limit the ranges of either variable initially. But as the regression continues, the ranges for the variables become more constrained and each must be updated to reflect this. For example, if before the equal constraint is encountered the ranges are $2 \leq \alpha_1 \leq 10$ and $5 \leq \alpha_2 \leq 15$, then after the constraint ρ for each variable will be (5 10). If later, one of the variables is further constrained, the other must be updated to reflect this. This is beyond the capabilities of the standard unification algorithm, because the variables are not bound to specific instances of expressions but ranges.

The algorithm in Figure 5.7 is used to compute the possible decision variable values. The algorithm computes all the possible values by determining which rules are applicable to a subgoal and traversing them to get the bindings. Back-Extend is the procedure that computes the ranges and constraints for the variables.

For Back-Extend to work properly, the procedure must understand the semantics of the primitive predicates. These semantics can either be built into the procedure or be accessible from the primitive predicate names. For example, the algorithm must know how to modify the ranges for a variable when a subgoal of (< ?x 5) is encountered. Figure 5.8 contains the Back-Extend algorithm. The details of restricting the ranges for the variables are not included in the algorithm but are straightforward. For example, an equal constraint sets ρ for a variable to the value of the constant in the goal. If that value is not in the current range for the variable, there exists a conflict in the rules used to get to this state so the process is aborted and no bindings are returned.

5.2.1 Extensions

The regression approach to determining variable value ranges is inefficient. One problem with the algorithm is that the system works through goals that may have no relation to the variable in question. One approach to help minimize the problem is to do a forward search[1] from the unsolved goal and determine which rules could possibly be effected by the value wanted.

[1] Forward in the sense of working from left to right in the partial-proof tree.

```
Compute-Possible-Bindings(G, Θ, Φ)
; G - the goal (subgoal) working on
; Θ - binding list from LHS of PPT
; Φ - possible values binding lists from RHS of PPT
PW = nil
Rules = set of rules with GΘ as head
    If Goal is on PPT border, only one rule applies
Rules = trim Rules to those that are determined to be usable
    See forward search in Section 5.2.1.
Rules = Rules minus the rules whose antecedents don't match with Θ and Φ
Do for each R in Rules
    Φ' = List(Copy(Φ))
    Do for each Subgoal in Antecedent(R)
        If Subgoal to the left of PPT border
            ignore
        Else
            NewΦ = nil
            Do for each φ in Φ'
                Push Back-Extend(Subgoal, Θ, φ) on NewΦ
            Enddo
            Φ' = NewΦ
        Endif
    Enddo
    Append Φ' to PW
Enddo
Return PW
```

Figure 5.7: Computing Possible Variable Values

During the forward search the rules that use the variable would be tagged. When regression is performed to compute the ranges, only the tagged rules are used. The extra work required by the forward search is beneficial if there are lots of branches to the right of the unsolved subgoal in the partial-proof tree. If all the branches are dependent upon the variable, then performing the forward search causes the proof tree to be traversed twice with little gain in speed for regression.

Another shortfall of the current approach is that the Φ bindings for ! FAIL rules are not handled. These rules insure certain combinations of bindings do not occur. One approach for

```
Back-Extend(Goal, Θ, φ)
; Goal – subgoal to be extended
; Θ - binding list for solved (left) portion of PPT
; φ - a single binding list for regression
If Goal is a chaining predicate
    Compute-Possible-Bindings(Goal. Θ. φ)
Else
    ; update the binding list φ
    Do for each variable v in Goal that is not in φ
        Create an entry for v, (v, total range for variable type, nil)
        Push entry on φ
    Enddo
    Case on predicate of Goal
    = or equal
        Update ρ (range) for variable(s) in Goal
        If both arguments to Goal are variables
            Push Goal on ν for each variable
        Endif
    < or > or <= or <=
        Update ρ (range) for variable(s) in Goal
        If both arguments to Goal are variables
            Push Goal on ν for each variable
        Endif
    ⋮       Continue with different effects on the range for each type of predicate
    Endcase
Return φ
```

Figure 5.8: Back-Extend

allowing those extensions is to add another component for the failure bindings to each entry on Φ. The problem is that failure bindings are not really constraints on the bindings of a particular variable. Instead ! FAIL rules constrain what bindings can occur between different variables (and their associated attributes). Currently such rules are ignored when computing the constraints for a variable. This means that the ranges produced could include values that will not allow a solution to be computed for some problems.

Another approach to solving this problem is to use a robust constraint propagation mechanism. For this to work, the system would have to construct the constraint network for each possible completion of the partial-proof tree. The constraint system would determine the value for the missing variable or determine if there exists an inconsistency. This approach could get better approximations for the missing variable value since all constraints would be taken into account. Unfortunately, it is expensive computationally to establish the network for each possible completion of the PPT.

5.3 Inducing Rule Consequents

After the value ranges for the missing variable have been computed using the regression technique, the consequent for the new rule must be induced. The system knows what variable is missing a value and what must be in the consequent. The formula for determining the value for the missing variable now needs to be discovered. For example, if the unsolved goal is `(foo ?x ?y ?z)` with ?y and ?z being bound to values, the system must find a rule of the form:

`(foo ?x ?y ?z)` ← conditions for rule `(=` `?x` *formula*`)`

The so called consequent of the rule is the induced *formula* that assigns a value to `?x`. It will be called the consequent in this discussion because assigning `?x` a value is the reason for firing the rule.

KBEG found these formulas in KBLØ. The situation is more complex in KBL1 because a range of values must be handled instead of a single number. Finding a specific answer for the variable is more difficult because an exact difference between a value produced by a formula and values for the variable in question (from other examples) can no longer be used to guide the search as was done in KBEG. The approach used, however, is similar to KBEG, with the system now trying to find formulas that contain ranges. The new approach is called KBEG-R.

KBEG-R is a simple system that does a search for formulas starting with constants and then adding variables to get more complex formulas. It does not use the notion of proportionality like ABACUS or KBEG, but instead the search is guided by finding *simple* formulas to cover the events. To help constrain the search, the user can specify the most complex formulas allowed, or even the specific forms of allowed formulas. The system manipulates ranges of new variables

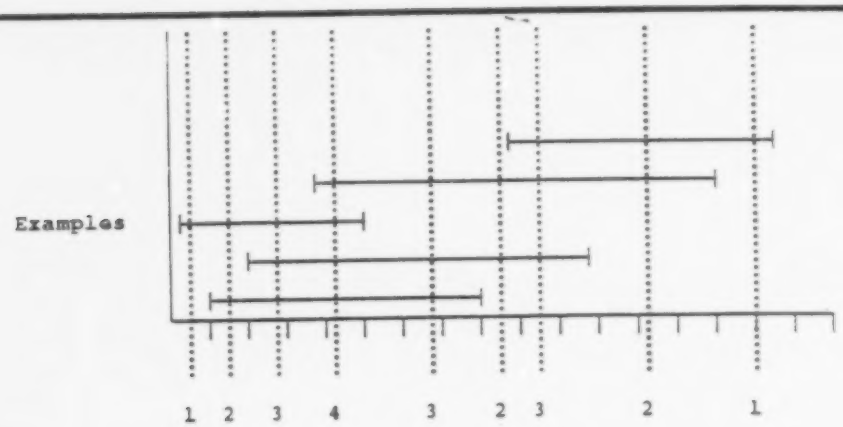


Figure 5.9: Determining Constant Value in KBEG-R

(that have been assigned the values of formulas) when computing new variables. The system stops when a constant value has been found for a subset of the points. Figure 5.9 demonstrates how this is done. If all the value ranges for the examples are plotted on a number line, the system can determine the number of ranges that include any point on the line. KBEG-R selects the interval intersecting the most examples (like KBEG, it must handle a specified minimum number). The system assumes that the mid-point of the interval is the constant value of the variable. In the example, the interval intersecting four examples is selected.

As a more complex example of how KBEG-R works, consider the data provided in Table 5.1. Each example has a bound variable, Y, and a computed range, X. When the system computes $X - Y$ a new range is computed. From these new ranges the system can determine the formula covering the examples is $X - Y = (2\ 3)$. The exact value for the formula is not determined in this case, but from this expression a value for X can be computed. To get the specific value for the equation, an example can be tested to check if the new constraint will allow a single value to be determined. More examples might allow an exact value to be used, or the system could assign any of the values.

5.4 Inducing Rule Antecedents

Before rules can be added to the knowledge base, they must contain antecedents that appropriately limit the scope of the rule. The rule should apply to all those cases explicitly found

Y	X	X − Y
0	(1 5)	(1 5)
1	(1 5)	(0 4)
2	(1 5)	(-1 3)
3	(5 10)	(2 7)
4	(6 8)	(2 4)
5	(7 9)	(2 4)
6	(1 100)	(-5 94)

Table 5.1: KBEG-R Example

through the collection of examples and partitioning. The KBLØ system used the INDUCE module to find the appropriate antecedents. Unfortunately, the rules found by INDUCE and other discriminant rule finding systems do not produce rules that engineers like to use. Engineers prefer rules that limit the values of many attributes. This constrains the applicability of the rule and conveys more information to the engineer.

In this section, the module producing rule antecedents for KBL1 is described. The system produces maximally specific generalizations (MSG). This system takes a heuristic approach to finding the MSG for a set of examples. Two sets of examples are presented to the system, a set to be described (positive) and a negative set. The descriptions produced should not cover any events from the negative set, if possible. Before describing the approach used in KBL1, some related systems are discussed. If these systems were readily available they could have been used. If a user of KBL1 would prefer not to have MSG descriptions for the events, some other rule induction system could be used with the other components of the system.

5.4.1 Related Work

There are a number of systems that have been written to produce MSG descriptions. Two systems are described here and contrasted with the approach implemented for KBL1. The two systems described are INDUCE 1.2 [Larson, 1977] and Thoth [Vere, 1980, Vere, 1977].

5.4.1.1 INDUCE 1.2

The INDUCE 1.2 system by Larson (also described in [Dietterich and Michalski, 1983]) can be used to find maximally specific generalizations. This system divides the search space

into two spaces. INDUCE first searches through a *structure-only space*. This space contains the non-unary predicates, e.g., *on-top*. The system finds plausible explanations using the structure-only space and searches the second space, *attribute space* to fill in the details of the description found.

The INDUCE algorithm uses a graph representation and graph match to find the MSG, where the system used for KBL1 uses literals and unification to find the MSG. INDUCE can also find descriptions that are not maximally specific. This occurs because not all MSG descriptions occur when the structural component is maximally specific. To allow the system to find these descriptions, INDUCE allows some nonmaximally specific structure-only descriptions to be used in the attribute space search.

5.4.1.2 Thoth

The approach used by Thoth is similar to the system developed for KBL1. Both systems generalize a description given a new event. Thoth first computes all *matching pairs* between the current description and new event. A pair of literals match when they have the same predicate and the same number of arguments. The system then selects subsets of the matching pairs so that no element in a pair occurs more than once. The system adds pairs of literals that have a matching argument (the predicates need not be the same). The system generalizes the sets of literals by turning constants to variables.

The largest drawback of the Thoth system is that only two generalization operators are used: dropping attributes and turning constants into variables. The system used by KBL1 generalizes by allowing internal disjunction and closing of intervals in addition to the operators used by Thoth.

The Thoth system does allow *negative existential quantification* over parts of the description. This is accomplished with the COUNTERFACTUALS module which works with the residual of an event. The residual of an event is the set of literals not included in a covering description. Thoth finds the not conditions by running the algorithm with the positive events consisting of the residuals from the (original) negative events and the negative events are the residuals from the (original) positive events. This recursion can occur to any desired depth. The results returned from a recursive call have a *not* placed in front of them. In this way, Thoth is able to construct descriptions that would cover negative events without the added counterfactuals.

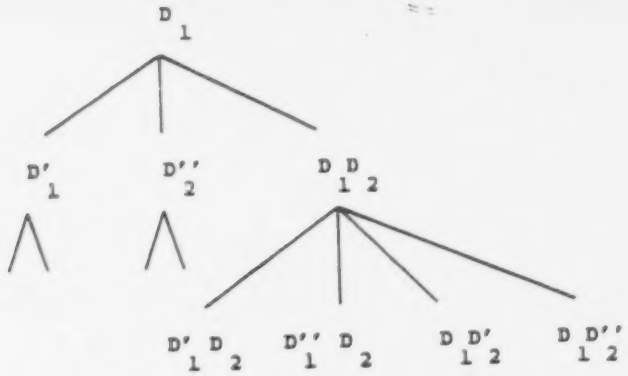


Figure 5.10: MSG Search

The system used by KBL1 does not handle this situation, but the methodology used by Thoth could be incorporated into the system.

5.4.2 KBL1's MSG

The induction module for KBL1 creates a most specific generalization for a set of positive examples by incrementally generalizing the current description with each event. A heuristic approach is used to limit the size of the search space. A beam search is used to limit the number of descriptions considered. Figure 5.10 will be used to help describe the process.

Each node of the search tree used to compute the MSG is a disjunction of descriptions. A description is a conjunction of literals. The branches of the tree are created by different generalizations to the description when including a new example. Each processed example adds a level to the tree. As shown in Figure 5.10, the initial description D_1 could be generalized in three ways to cover example E_1. The first two extensions involve generalizing the existing description to get D_1' and D_1''. The third extension creates a new description D_2 to cover the event. Note that when a new description is created the old description does not get modified. When event $E2$ is introduced, there are a number of ways that the system can generalize the nodes.

The system uses four heuristics to determine which nodes are the best to expand. The system prefers nodes with the following characteristics:

- Few number of disjuncts.

- Few negative events covered (zero covered is desired).
- Descriptions with many *predicates*.
- Descriptions with few internal disjuncts.
- Descriptions that use as few variables as possible.

The user provides weights for each of these values and the node with the smallest value is considered the best. The formula used is:

$$\text{number-disjuncts} * W_1 + \text{number-negs-covered} * W_2 - \text{number-predicates} * W_3 + \text{number-internal-disjuncts} * W_4 + \text{number-variables-used} * W_5$$

The previous paragraphs describe how the search through descriptions is conducted. Figure 5.11 contains the algorithm used to actually compute the description. The algorithm is provided with a description (MSG of previous events), an event that should be covered by the description, a binding list, and partially complete new description. The algorithm produces all the ways that the description could be generalized to cover the new event. The partial description contains the modified components from the previous matches. The binding list maintains the variable bindings required to keep the description as specific as possible.

The MSG routine produces all the descriptions; it is the responsibility of the search mechanism to trim the nodes that are not likely to be productive. If there are multiple descriptions in a node, the MSG algorithm is just run for each description. The results are concatenated with the unmodified descriptions.

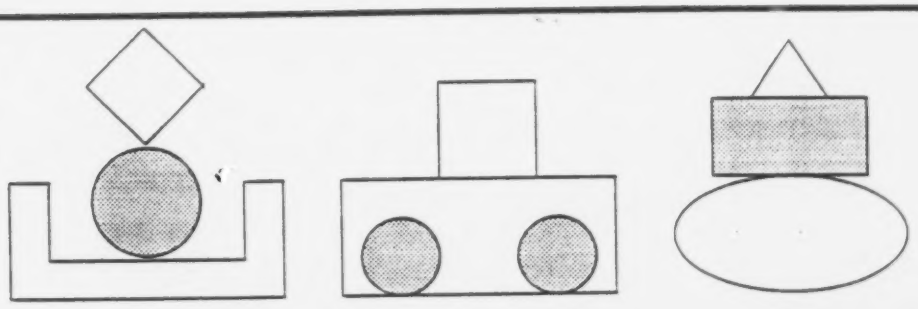
To compare MSG with other algorithms that produce characteristic descriptions, the examples used in [Dietterich and Michalski, 1983] were run on the system. Figure 5.12 shows the figures used along with their descriptions. The MSG algorithm produces many descriptions for the three events. The following description was returned as the best:

```
((SHAPE ?X <(TRIANGLE or SQUARE or DIAMOND)>) (TEXTURE ?X <(BLANK)>)
 (SIZE ?X <(MEDIUM)>) (SHAPE ?Y <(RECTANGLE or CIRCLE)>)
 (TEXTURE ?Y <(BLANK or SHADED)>) (SIZE ?Y <(LARGE or MEDIUM)>)
 (SHAPE ?Z <(ELLIPSE or CIRCLE or BOX)>) (TEXTURE ?Z <(SHADED or BLANK)>)
 (SIZE ?Z <(SMALL or LARGE)>) (ONTOP ?X ?Y))
```

```
MSG(description, event, Θ, PD)
; description - conjunction of literals with variables
; event - conjunction of grounded literals describing the event
; Θ - binding list
; PD - partial generalized description
If (Null description)
    Return(PD)
Else
    Literals = all the literals in event that unify with the first clause of the description given Θ
    If (Null Literals)
        Return(MSG(tail(description), event, Θ, PD)
    Else
        Do for each L in Literals
            Θ′ = Unify(L, Head(description), Θ)
            If (Null Θ′) ;no unification
                Create new variables in Head(description) to cover different values
            Endif
            PD2 = PD ∪ SpGen(L, Head(description))
            R = MSG(Tail(description), event, Θ′, PD2)
            Push R onto Results
        Enddo
        Return(Results)
    Endif
Endif

SpGen(lit, desc)
; lit - a literal from the event
; desc - a clause from the description
If Covers(desc, lit)
    Return(desc)
Else
    New-desc = union lit values with desc
    Return(New-desc)
```

Figure 5.11: MSG Description Generation



E1 ((shape v1 diamond) (texture v1 blank) (size v1 medium) (shape v2 circle) (texture v2 shaded) (size v2 medium) (shape v3 box) (texture v3 blank) (size v3 large) (ontop v1 v2) (ontop v2 v3)))

E2 ((shape v1 square) (texture v1 blank) (size v1 medium) (shape v2 circle) (texture v2 shaded) (size v2 small) (shape v4 circle) (texture v4 shaded) (size v4 small) (shape v3 rectangle) (texture v3 blank) (size v3 large) (ontop v1 v3) (inside v2 v3) (inside v4 v3)))

E3 ((shape v1 triangle) (texture v1 blank) (size v1 medium) (shape v2 rectangle) (texture v2 shaded) (size v2 medium) (shape v3 ellipse) (texture v3 blank) (size v3 large) (ontop v1 v2) (ontop v2 v3)))

Figure 5.12: Examples for MSG

INDUCE produced the following description for the three events:

```
((ONTOP ?X ?Y) (SIZE ?X MEDIUM)(SHAPE ?X POLYGON) (TEXTURE ?X BLANK)
 (SIZE ?Y <(MEDIUM or LARGE)>) (SHAPE ?Y <(RECTANGLE or CIRCLE)>))
```

There are differences between the results produced by the programs. First, MSG includes information about a third object in each of the events (?Z) and provides the details on those other objects. Second, MSG maintains the values of attributes when all the values have been seen. For example, (TEXTURE ?Y <(BLANK or SHADED)>) is maintained by MSG where INDUCE drops it from the description.

5.5 Specializing Created Rules

Although the KBL1 system takes special efforts to reduce the chances of producing overly general rules, it sometimes fails. One approach to correcting overly general rules is to find them and specialize them when the system is not working on solving problems. Examples can be selected at random from the example base and solved using the current knowledge base. If a solution different from the solution provided for the example is found, a rule–rule conflict has occurred. The specializer would then look through the proof tree for rules relating to the incorrect values. These rules can then be specialized so that the example will not match the rule's antecedents[2].

5.6 Converting AND/OR Trees to OR Trees

The rule-structured induction procedure presented previously was described as a problem-reduction (AND/OR tree) search. Since the search space is quite large, a working system will be required to use a heuristic approach in finding a solution. Most heuristic search algorithms are written for state-space (OR tree) search. As presented in [Barr and Feigenbaum, 1981], page 42, a problem-reduction search can be converted to state-space search by making each node of the state-space search tree contain a stack of goals and the current state. This has been done for RSI. The stack on each node contains the subgoals for solving a problem. The operators for the search pop the top subgoal from the stack and expand it. Expanding the subgoal is accomplished by finding rules that unify with the subgoal, matching it to an attribute of the problem, or determining a generalization to perform on the subgoal so that it will match an attribute. When a rule is found for the current subgoal, the antecedents of the rule are pushed onto the goal stack. The state saved on each node is the binding list computed from solving previous subgoals. When the search goal is an empty stack, the problem has been solved. When the problem is solved, the system needs to note the generalizations performed and apply those changes to the knowledge base.

Given a knowledge base with the rules:

$G \leftarrow A_1 A_2$

$G \leftarrow B_1 B_2$

[2]The specialization routine has not been implemented for this dissertation.

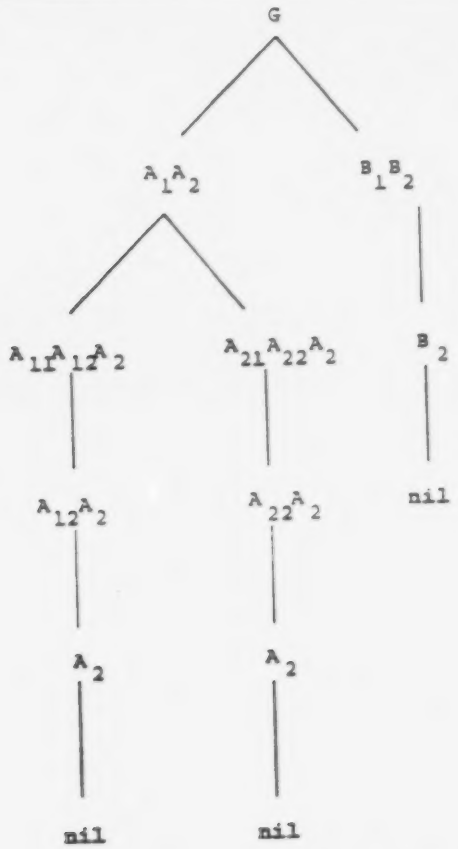


Figure 5.13: State-Space Example

$A_1 \leftarrow A_{11}A_{12}$

$A_1 \leftarrow A_{21}A_{22}$

and assuming that the subgoal is a primitive (i.e., there are no rules for the subgoal), then Figure 5.13 shows what the complete state-space tree would be. The binding list is not in the figure.

For the KBL1 system, the search is guided by the goal of finding a complete proof tree. When the RSI system is used, each node of the search tree is a partial-proof tree that contains the history of how this node in the search was completed. The search space actually being manipulated contains the rules used to complete the PPTs along with the search for the generalizations required to complete the proof. Figure 5.14 shows what the search tree (conceptually) is when each node contains a PPT.

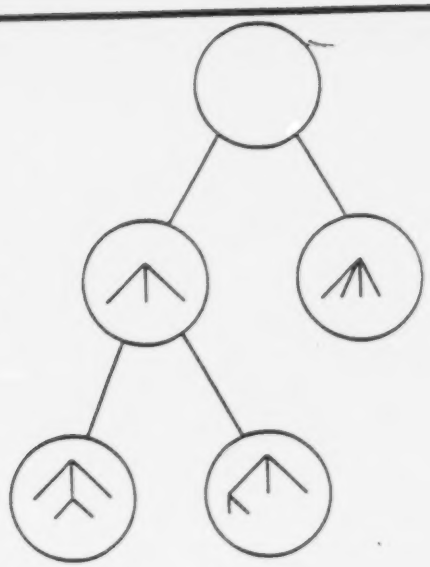


Figure 5.14: Search with PPT Nodes

5.7 KBL1

Until now, this chapter has described the components of KBL1: the rule-structured induction, the regression scheme used to determine values, KBEG-R, and the MSG module. In this section, the focus is on the interaction of these components when learning rules for the knowledge base. KBL1 uses the same basic approach as KBLØ:

1. Have a problem that is unsolvable using the existing knowledge base.
2. Construct a partial-proof tree for the problem.
3. Collect examples that match the partial-proof tree.
4. Repair the knowledge base so that the problem may be solved.

In the KBLØ system, the only repair to the knowledge base is the addition of new rules that are induced from examples. Constructing the PPT in KBLØ is straight forward because of the / (cut) operator. The KBL1 system has more options in repairing the knowledge base. Besides adding new rules, the system can generalize existing rules. This allows the system to induce conservative new rules available for later generalization as more knowledge is gained through

problem solving. Constructing the partial-proof tree is more complex in KBL1 because the rules no longer have the cut operators indicating where learning should occur[3].

KBL1 searches a space of possible solutions for the current, unsolved problem. Each node in the search tree contains a stack of the subgoals yet to be solved. The current state is maintained as a partial-proof tree and the binding list used to instantiate the tree for the problem. The operators for the search expand the top subgoal on the stack for the node. The operators are the same as those described in Section 5.1. During the heuristic search, the system performs as little generalization as possible. When there exists a perfect match, no other type of expansion is explored. The system will always attempt to extend a range before inducing a new rule.

When a new rule must be induced to complete the problem solution, the partial-proof tree is used to collect example/solution pairs from the example base. The system must then use the regression algorithm described earlier to get the values for the subgoal to be used as the consequent of the rule. The examples are now partitioned by KBEG-R into classes based upon those values and induction is performed. The induction performed by KBL1 is *most specific generalization*. In the KBLØ system, having the induction module produce most specific generalizations was not practical because the system did not contain a method for correcting overly specific rules. When RSI is used, over-specific rules can be added because those rules can later be generalized.

Potentially, a number of rules are generated for one subgoal in the partial-proof tree. This occurs when KBEG-R computes many possible value ranges (possible worlds). The system selects a single value from one of these ranges. This value is used to construct the binding portion of a new rule. The system then continues with the search for a proof (solution) to the problem. If at some point the problem cannot be solved with the selected value, the system will select a different range of values for some of the examples. This new range might allow the problem to be solved. If none of the induced rules can be used to extend the current problem, this branch of the search terminates without a solution. If all branches of the search are unsuccessful, then there does not exist enough knowledge in the example base to allow a

[3]For efficiency, the user might still want to have / operators. RSI can then work down from the / instead of from the top-level goal. In the rest of this section the RSI approach is described as occurring from the top most goal, this goal could be the top most goal identified with a / instead of the root of the proof tree.

```
(TOP ?X ?W) ← (A ?Y ?X) (B ?Z ?Y) (C ?W ?Z ?Y ?X)
(A ?Y ?X) ← (A1 ?X) (= ?Y (BINDING SLOT))
(A ?Y ?X) ← (A2 ?X) (= ?Y (BINDING HOLE))
(B ?Z ?Y) ← (B1 ?X) (< ?X 5) (= ?Z (BINDING NARROW))
(B ?Z ?Y) ← (B1 ?X) (≥ ?X 5) (= ?Z (BINDING WIDE))
(C ?W NARROW HOLE ?X) ← (= ?W (BINDING DRILL))
(C ?W WIDE HOLE ?X) ← (= ?W (BINDING 2-PASS-DRILL))
(C ?W WIDE SLOT ?X) ← (= ?W (BINDING 2-PASS-END-MILL))
(C ?W NARROW SLOT ?X) ← (= ?W (BINDING END-MILL))
```

Figure 5.15: Toy CAPP Knowledge Base

conversion from that implicit knowledge into the explicit rule knowledge needed for solving the problem.

5.8 Demonstration of KBL1

The demonstrations of the KBL1 system in this section illustrate how the components of the system work. Empirical evaluation of KBL1 is provided in the next chapter and is not discussed here. Each of the following subsections presents a single demonstration of KBL1 in use. The sections contain the actual data used to run these examples with minimal changes made for clarification purposes.

All the demonstrations use rules from a toy CAPP (Computer-Aided Process Planning) domain. The top-level goal is TOP and the idea is to determine the type of machining required for the object provided. The two variables to TOP are bound to the type of feature found, and the machining needed to produce that feature. The complete knowledge base is in Figure 5.15. The predicates A1 and A2 have unusual semantics in this knowledge base. They are propositional in the sense that their argument has no meaning[4]. The presence of A1 indicates a slot is being processed and A2 indicates that a hole should be machined. In a more complex domain theory, subgoal A would be used to obtain the type of feature, instead of just having propositional values indicate the appropriate type. The subgoal B is used to determine the size of the feature.

[4]The system recognizes that the argument has no meaning since it is defined in the language specification as a nominal type with NIL values. This means any value is accepted and the value is ignored by the system.

either narrow or wide. The subgoal C determines what the actual processing must be. drilling or end-milling, and if a one or two pass process is required. The = predicate is used to set the first argument to the value of the second argument. The second argument is evaluated and BINDING just returns the value of its argument (similar to quote in LISP).

5.8.1 Single Rule Constructed

In this section, the system is demonstrated on a problem that requires one rule to be induced. KBL1 requires four pieces of information in order to work: language specification, examples, knowledge base, and an unsolvable problem. The language specification used for the first example defines all the values as nominals or linears. The language definition used is:

```
(LANG-SPEC TOP (FEATURE :NOMINAL NIL) (VALUE :NOMINAL (DRILL 2-PASS-DRILL
   END-MILL 2-PASS-END-MILL)))
(LANG-SPEC A (VALUE :NOMINAL (SLOT HOLE)) (FEATURE :NOMINAL NIL))
(LANG-SPEC A1 (VALUE :NOMINAL NIL))
(LANG-SPEC A2 (VALUE :NOMINAL NIL))
(LANG-SPEC B (VALUE :NOMINAL (NARROW WIDE)) (FEATURE :NOMINAL (SLOT HOLE)))
(LANG-SPEC B1 (VALUE :LINEAR (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 20
   25)))
(LANG-SPEC C (VALUE :NOMINAL (DRILL 2-PASS-DRILL END-MILL 2-PASS-END-MILL))
   (WIDTH :NOMINAL (NARROW WIDE)) (TYPE :NOMINAL (SLOT HOLE)) (FEATURE
   :NOMINAL NIL))
```

The special name of *FEATURE* is used to indicate that the variable will be bound to an object name. The learning system should therefore not use this argument in the construction of new rules, but it is used to maintain binding constraints.

Seven examples were stored in the example base for this problem. The fact/solution pairs for each example are:

1. (A1 SLOT-1) (B1 3) / (TOP SLOT-1 END-MILL)

2. (A1 SLOT-2) (B1 2) / (TOP SLOT-2 END-MILL)

3. (A1 SLOT-3) (B1 4) / (TOP SLOT-3 END-MILL)

4. (A1 SLOT-4) (B1 6) / (TOP SLOT-4 2-PASS-END-MILL)

5. (A1 SLOT-5) (B1 10) / (TOP SLOT-5 2-PASS-END-MILL)

6. (A2 HOLE-1) (B1 3) / (TOP HOLE-1 DRILL)

7. (A2 HOLE-2) (B1 10) / (TOP HOLE-2 2-PASS-DRILL)

The facts for the problem, that is the values of the problem to be solved are: (A1 SLOT-E) (B1 4). The rule

(C ?W NARROW SLOT ?X) ← (= ?W (BINDING END-MILL))

was removed from the knowledge base (Figure 5.15) in order to have an incomplete domain theory during the run of the system. All the required components for a run of the system have now been defined. When presented with the query (TOP ?X ?Y), KBL1 attempts to find a solution for the given facts.

The system determined that the best partial-proof tree to use in the induction of new rule was:

```
Bindings:
(T (?Z1 NARROW) (?X2 4) (?Y2 ?Y1) (?Z2 ?Z1) (?Y1 SLOT)
   (?X4 SLOT-E) (?X3 ?X1) (?Y3 ?Y1) (?W2 ?W1) (?X1 ?X4))
Proof:
(RULE-PROOF (TOP ?X1 ?W2)
 (((A ?Y1 ?X1)
   (RULE-PROOF (A ?Y3 ?X3)
    (((A1 ?X3) (A1 SLOT-E))
     ((= ?Y3 (BINDING SLOT)) (= SLOT (BINDING SLOT))))))
  ((B ?Z1 ?Y1)
   (RULE-PROOF (B ?Z2 ?Y2)
    (((B1 ?X2) (B1 4))
     ((< ?X2 5) (< 4 5))
     ((= ?Z2 (BINDING NARROW)) (= NARROW (BINDING NARROW))))))
  ((C ?W2 ?Z1 ?Y1 ?X1) (MADE-UP (C ?W1 NARROW SLOT SLOT-E)))))
```

In the proof structure, each goal has a RULE-PROOF label. The next item in the list is the head of the rule. This is followed by the antecedents for the rule. Following each antecedent (nested) is the proof used to prove it. If a ground fact was used, no RULE-PROOF label is given as it indicates chaining occurred. If a goal could not be solved, the MADE-UP label is placed around the hypothesized solution. The first MADE-UP form indicates where a rule needs to be learned.

Using this partial-proof tree the system collected three examples, 1, 2, and 3, that match the context. All three examples were placed in the same partition. Using those examples the system constructed the rule:

```
(C ?W ?Z ?Y ?X) ← (A1 ?U) (B1 ?V) (MEMBER ?V (3 2 4))
        (= ?W (BINDING END-MILL))
```

Using the induced rule, KBL1 was able to complete the proof for the problem.

```
Binding:
(T (?W3 END-MILL) (?X1 4) (?X2 SLOT-E) (?X3 SLOT-E) (?Y1 SLOT)
   (?Z1 ?Z2) (?W1 ?W3) (?Z2 NARROW) (?X5 4) (?Y3 ?Y2) (?Z3 ?Z2)
   (?Y2 SLOT) (?X7 SLOT-E) (?X6 ?X4) (?Y4 ?Y2) (?W2 ?W3) (?X4 ?X7))
Proof:
(RULE-PROOF (TOP ?X4 ?W2)
 (((A ?Y2 ?X4)
  (RULE-PROOF (A ?Y4 ?X6)
   (((A1 ?X6) (A1 SLOT-E))
    ((= ?Y4 (BINDING SLOT)) (= SLOT (BINDING SLOT))))))
   ((B ?Z2 ?Y2)
    (RULE-PROOF (B ?Z3 ?Y3)
     (((B1 ?X5) (B1 4))
      ((< ?X5 5) (< 4 5))
      ((= ?Z3 (BINDING NARROW)) (= NARROW (BINDING NARROW))))))
   ((C ?W2 ?Z2 ?Y2 ?X4)
    (RULE-PROOF (C ?W1 ?Z1 ?Y1 ?X3)
     (((A1 ?X2) (A1 SLOT-E))
      ((B1 ?X1) (B1 4))
```

```
((MEMBER ?X1 (3 2 4)) (MEMBER 4 (3 2 4)))
((= ?W1 (BINDING END-MILL)) (= END-MILL (BINDING END-MILL))))))))
```

There are a couple of points to notice about the produced rule. First, the system includes the fact that A1 is present in all the examples. This is how the system can determine that the machined feature is a slot. Also, since the variables were defined as nominals the value of B1 is checked using a MEMBER function instead of numeric tests. The range is not as large as in the original rules because the current examples did not cover the whole range. This range can be extended as future problems are encountered by the system. The third demonstration shows such a case.

5.8.2 Two Rules Constructed

For this demonstration, two rules need to be induced in order to complete the solution for the problem. The language specification has been modified so that real values are now used instead of nominals for the B1 predicate. The language specification is:

```
(LANG-SPEC TOP (FEATURE :NOMINAL NIL) (VALUE :NOMINAL (DRILL 2-PASS-DRILL
   END-MILL 2-PASS-END-MILL)))
(LANG-SPEC A (VALUE :NOMINAL (SLOT HOLE)) (FEATURE :NOMINAL NIL))
(LANG-SPEC A1 (VALUE :NOMINAL NIL))
(LANG-SPEC A2 (VALUE :NOMINAL NIL))
(LANG-SPEC B (VALUE :NOMINAL (NARROW WIDE)) (FEATURE :NOMINAL (SLOT HOLE)))
(LANG-SPEC B1 (VALUE :REAL 0 30))
(LANG-SPEC C (VALUE :NOMINAL (DRILL 2-PASS-DRILL END-MILL 2-PASS-END-MILL))
   (WIDTH :NOMINAL (NARROW WIDE)) (TYPE :NOMINAL (SLOT HOLE)) (FEATURE
   :NOMINAL NIL))
```

The examples stored in the example base are the following:

```
1. (A1 SLOT-1) (B1 3) / (TOP SLOT-1 END-MILL)

2. (A1 SLOT-2) (B1 2) / (TOP SLOT-2 END-MILL)

3. (A1 SLOT-3) (B1 4) / (TOP SLOT-3 END-MILL)
```

4. (A1 SLOT-3A) (B1 4) / (TOP SLOT-3A END-MILL)

5. (A1 SLOT-3B) (B1 4) / (TOP SLOT-3B END-MILL)

6. (A1 SLOT-4) (B1 6) / (TOP SLOT-4 2-PASS-END-MILL)

7. (A1 SLOT-5) (B1 10) / (TOP SLOT-5 2-PASS-END-MILL)

8. (A2 HOLE-1) (B1 3) / (TOP HOLE-1 DRILL)

9. (A2 HOLE-2) (B1 10) / (TOP HOLE-2 2-PASS-DRILL)

The initial facts provided are the same as in Section 5.8.1, (A1 SLOT-E) (B1 4). The following two rules were removed from the complete domain theory of Figure 5.15:

(A ?Y ?X) ← (A1 ?X) (= ?Y (BINDING SLOT))

(C ?W NARROW SLOT ?X) ← (= ?W (BINDING END-MILL))

While attempting to solve this problem, the system fails and constructs the partial-proof tree:

Bindings:

(T (?W2 ?W1) (?X2 ?X1))

Proof:

(RULE-PROOF (TOP ?X2 ?W2)
 (((A ?Y1 ?X2) (MADE-UP (A ?Y1 ?X1)))
  ((B ?Z1 ?Y1) (MADE-UP (B ?Z1 ?Y1)))
  ((C ?W2 ?Z1 ?Y1 ?X2) (MADE-UP (C ?W1 ?Z1 ?Y1 ?X1)))))

All nine of the examples were collected to help the system induce a rule for subgoal A. But because a rule for C was also missing, the system could not derive solutions for all the examples using the regression algorithm described in Section 5.2. Solutions could only be found for examples 6, 7, 8, and 9 so they were used to induce the rule:

(A ?Y1 ?X3) ← (A1 ?X2) (B1 ?X1) (≤ 4 ?X1) (≤ ?X1 10) (= ?Y1 (BINDING SLOT))

With the new rule, KBL1 continues working to find a solution for the problem. KBL1 stops back chaining when a rule was needed for C. The system uses the following partial-proof tree to collect examples for rule induction:

```
Bindings:
(T (?Z1 NARROW) (?X3 4) (?Y2 ?Y1) (?Z2 ?Z1) (?Y1 SLOT)
   (?X4 4) (?X5 SLOT-E) (?X6 ?X1) (?Y3 ?Y1) (?W2 ?W1) (?X2 ?X1))
Proof:
(RULE-PROOF (TOP ?X2 ?W2)
 (((A ?Y1 ?X2)
   (RULE-PROOF (A ?Y3 ?X6)
    (((A1 ?X5) (A1 SLOT-E))
     ((B1 ?X4) (B1 4))
     ((<= 4 ?X4) (<= 4 4))
     ((<= ?X4 10) (<= 4 10))
     ((= ?Y3 (BINDING SLOT)) (= SLOT (BINDING SLOT))))))
  ((B ?Z1 ?Y1)
   (RULE-PROOF (B ?Z2 ?Y2)
    (((B1 ?X3) (B1 4))
     ((< ?X3 5) (< 4 5))
     ((= ?Z2 (BINDING NARROW)) (= NARROW (BINDING NARROW))))))
  ((C ?W2 ?Z1 ?Y1 ?X2) (MADE-UP (C ?W1 NARROW SLOT ?X1)))))
```

The examples 3, 4, and 5 were matched by this partial-proof tree and used to induce the rule:

$$(C\ ?W1\ ?Z1\ ?Y1\ ?X3) \leftarrow (A1\ ?X2)\ (B1\ ?X1)\ (\leq 4\ ?X1)\ (\leq ?X1\ 4)\ (= ?W1\ (\text{BINDING END-MILL}))$$

With the second rule, the system is able to complete the proof for the problem.

5.8.3 Extending an Existing Rule

KBL1 produces rules with antecedents that are maximally specific generalizations from a set of examples. As could be seen in the two previous demonstrations, many rules created are over specific. KBL1 can generalize existing rules in the system when such generalization allows for the completion of a proof. This is now demonstrated.

The language specification for this problem is almost the same as that presented for the previous demonstration. One modification has occurred for B1. The new specification is:

(LANG-SPEC B1 (VALUE :REAL 0 30 2))

where the 2 is the tolerance that is allowed when extending this value. The tolerance is the maximum value change that that can occur in a rule, and still have the change be considered a slight-extend. Larger values than the tolerance can be extended on a rule, but then an extreme-extend factor is used by the RSI module in determining the cost of the extension.

Let the knowledge base (Figure 5.15) contain the rule

```
(B ?Z ?Y) ← (B1 ?X) (< ?X 3) (= ?Z (BINDING NARROW))
```

in place of

```
(B ?Z ?Y) ← (B1 ?X) (< ?X 5) (= ?Z (BINDING NARROW)).
```

The example base is provided with the same examples as in Section 5.8.1 (though it does not matter in this case because the RSI module performs the induction without looking at the examples). When the system is run with the facts used previously, the rule

```
(B ?Z ?Y) ← (B1 ?X) (≤ ?X 4) (= ?Z (BINDING NARROW))
```

is created to cover the example. The extension is within the tolerance defined by the language specification, so this is considered a minor change to the knowledge base. The rule is still not correct, because it needs to be more general (allow values less than 5). RSI only generalizes the rule enough to cover the current example.

5.9 KBL1 Summary

KBL1 is one example of a system that exploits the KBL theory. The system combines knowledge from an example base and a knowledge base to induce rules missing from the initial knowledge base. The system incorporates heuristic search in the construction of the partial-proof trees and in constructing the most specific generalizations. By modifying the search heuristics, the system can be tuned to perform for different domains.

Constructing new rules, using existing knowledge and examples is computationally expensive. The system produces many different options for various forms of knowledge, e.g. partial-proof trees, possible worlds for variable bindings, and MSG descriptions. Combinatorial explosion occurs when these forms of information can be combined to find new rules. The KBL1 implementation has been able to perform by severely restricting the possible combinations ac-

tually pursued. Simple mechanism have been implemented, but the KBL methodology allows for much more complex mechanisms to be used advantageously.

Chapter 6

Empirical Results

This chapter presents results of testing the KBL1 system. Through descriptions of experimental results, the capabilities of the KBL1 system, and therefore the knowledge-based learning paradigm, will be explored. The KBL1 system is not directly compared with other learning systems. A comparison of KBLØ to ID3 was presented in Chapter 4. Instead, the results presented in this chapter explain the usefulness of different knowledge types in rule induction. Issues such as "How does having a larger example base effect rule learning?" and "Does a larger partial-proof tree improve the accuracy of induced rules?" are addressed.

In the next section, a set of experiments is presented that characterizes three important aspects of the KBL approach. The second section describes an experiment that illustrates one of the points, using a larger domain theory.

6.1 Justifying KBL

There are three important points that will be made through the presentation of experiments in this section:

- **Rules induced with examples selected via a context (from a partial-proof tree) perform better than rules induced with all examples.** In other words, KBL1 produces rules that are more appropriate for use by the performance element than pure empirical techniques.

- **KBL1 performs better when more examples are presented.** The system is empirical in nature; the rules are created inductively. The performance of the system can be enhanced through the addition of new examples, as with any similarity-based system.

- **The usefulness of a partial-proof tree in defining the appropriate context for rule induction is domain dependent.** Contexts improve learning when the outcome of a subgoal depends significantly on the attributes used in solving previous subgoals. Determining the correct rule to use in solving a subgoal is *dependent* upon the previous rules fired. Contexts are less useful when rules are *independent* of values used in previously fired rules.

There are two domain theories used in this experiment. Each domain theory works on events with four attributes. Each attribute has four possible values, `a`, `b`, `c`, and `d`, which defines a problem space of 256 total events. The top-level goal, `(output ?x ?v1 ?v2 ?v3)`, has 4 arguments, the object and three values. Each of the three values is defined as being a nominal with values `a`, `b`, `c`, and `d`. The *dependent* domain theory is given in Figure 6.1 and the *independent* domain theory is in Figure 6.2.

In the dependent domain theory, the value returned by the subgoal `find-r1` only depends upon the value of attribute `v1`. The value returned by `find-2` uses the value of `v2` and is dependent upon the value of `v1`. Notice that the important values of `v1` are `a` and `b`; the same values used for `find-r1`. The rules of `find-r3` define similar dependencies to `find-r1` and `find-r2`. This dependency is important when the partial-proof tree is used to define the context for rule induction. The context defined by previously fired rules will have *relevance* to the current problem.

The independent domain theory uses the same subgoals as the dependent theory. However, the values returned (bindings created) by the rules for one subgoal are completely independent from previously computed values.

6.1.1 Experimental Results

Each domain theory was tested with three different sized example bases of 25, 50, and 75 events. Each test consisted of removing a single rule from the domain theory and randomly selecting events from the problem space for the example base. A test problem was selected from

```
1. (output ?x ?y ?z ?w) ← (find-r1 ?x ?y) (find-r2 ?x ?z) (find-r3 ?x ?w)

2. (find-r1 ?x ?y) ← (v1 ?x ?z) (member ?z (a b)) (= ?y (binding a))

3. (find-r1 ?x ?y) ← (v1 ?x c) (= ?y (binding b))

4. (find-r1 ?x ?y) ← (v1 ?x d) (= ?y (binding b))

5. (find-r2 ?x ?y) ← (v1 ?x ?z) (member ?z (a b)) (v2 ?x ?w)
   (member ?w (a b)) (= ?y (binding a))

6. (find-r2 ?x ?y) ← (v1 ?x ?z) (member ?z (c d))(v2 ?x ?w)
   (member ?w (a b)) (= ?y (binding b))

7. (find-r2 ?x ?y) ← (v1 ?x ?z) (member ?z (a b))(v2 ?x ?w)
   (member ?w (c d)) (= ?y (binding c))

8. (find-r2 ?x ?y) ← (v1 ?x ?z) (member ?z (c d))(v2 ?x ?w)
   (member ?w (c d)) (= ?y (binding d))

9. (find-r3 ?x ?y) ← (v1 ?x ?a) (member ?a (a b)) (v2 ?x ?b)
   (member ?b (a b)) (v3 ?x ?c) (member ?c (a b)) (v4 ?x ?d)
   (member ?d (a b)) (= ?y (binding a))

10. (find-r3 ?x ?y) ← (v1 ?x ?a) (member ?a (c d)) (v2 ?x ?b) (member ?b (a
    b)) (v3 ?x ?c) (member ?c (a b)) (v4 ?x ?d) (member ?d (a b))
    (= ?y (binding b))

11. (find-r3 ?x ?y) ← (v1 ?x ?a) (member ?a (c d)) (v2 ?x ?b)
    (member ?b (c d)) (v3 ?x ?c) (member ?c (c d)) (v4 ?x ?d)
    (member ?d (c d)) (= ?y (binding c))

12. (find-r3 ?x ?y) ← (v1 ?x ?a) (member ?a (a b)) (v2 ?x ?b)
    (member ?b (c d)) (v3 ?x ?c) (member ?c (c d)) (v4 ?x ?d)
    (member ?d (c d)) (= ?y (binding d))

13. (find-r3 ?x ?y) ← (v1 ?x ?a) (member ?a (a b)) (v2 ?x ?b)
    (member ?b (c d)) (v3 ?x ?c) (member ?c (c d)) (v4 ?x ?d)
    (member ?d (c d)) (= ?y (binding d))

14. (find-r3 ?x ?y) ← (= ?y (binding d))
```

Figure 6.1: Dependent Domain Theory

```
1. (output ?x ?y ?z ?w) ← (find-r1 ?x ?y) (find-r2 ?x ?z) (find-r3 ?x ?w)

2. (find-r1 ?x ?y) ← (v1 ?x ?z) (member ?z (a b)) (= ?y (binding a))

3. (find-r1 ?x ?y) ← (v1 ?x c) (= ?y (binding b))

4. (find-r1 ?x ?y) ← (v1 ?x d) (= ?y (binding b))

5. (find-r2 ?x ?y) ← (v2 ?x ?z) (member ?z (a b)) (= ?y (binding a))

6. (find-r2 ?x ?y) ← (v2 ?x c) (= ?y (binding b))

7. (find-r2 ?x ?y) ← (v2 ?x d) (= ?y (binding b))

8. (find-r3 ?x ?y) ← (v3 ?x ?z) (member ?z (a b))(v4 ?x ?w)
   (member ?w (a b))(= ?y (binding a))

9. (find-r3 ?x ?y) ← (v3 ?x ?z) (member ?z (a b)) (= ?y (binding b))

10. (find-r3 ?x ?y) ← (v4 ?x ?z)(member ?z (a b)) (= ?y (binding c))

11. (find-r3 ?x ?y) ← (= ?y (binding d))
```

Figure 6.2: Independent Domain Theory

the problem space and presented to the system. If the problem could not be solved, a rule was learned using the context defined by a partial solution. For each removed rule and example base size, twenty such tests were performed. After a rule was learned and added to the domain theory, solutions for all points in the problem space were computed. The solution for an event was compared with the correct solution.

The results of the tests for each size example base are presented in two ways, %-difference and %-improvement. %-difference is defined as $\frac{N-O}{O}$, where N is the number of events solved correctly by the new domain theory (after adding induced rule) and O is the number of events the incomplete theory solved correctly. %-difference is plotted for the number of constraints in the context in Figures 6.3, 6.4, and 6.5.

%-improvement is defined as $\frac{N-O}{P}$ where $N$and O retain their previous definitions, and P is the size of the problem space. These results are shown in Figures 6.6, 6.7, and 6.8.

Comparing the graphs for different example base sizes shows that KBL1 does produce better rules when more examples are present. This demonstrates the empirical nature of the system.

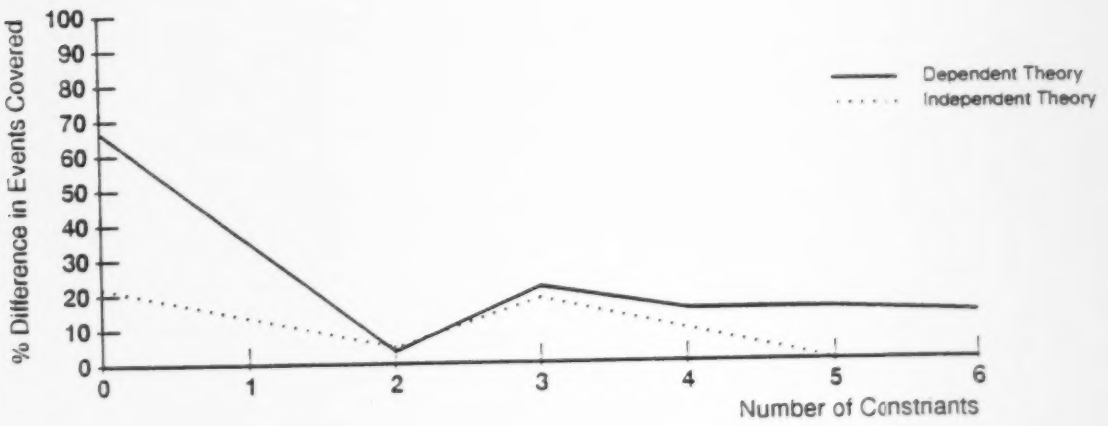


Figure 6.3: Example Base with 25 Events

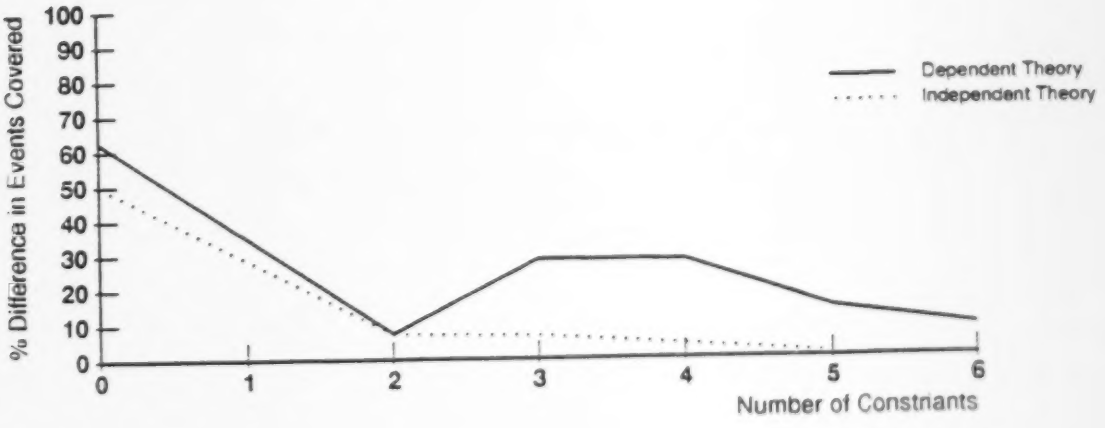


Figure 6.4: Example Base with 50 Events

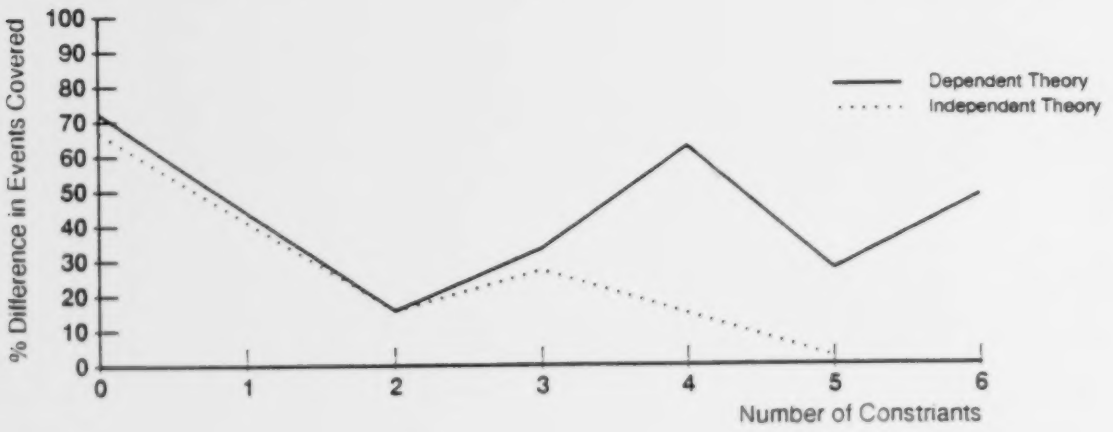


Figure 6.5: Example Base with 75 Events

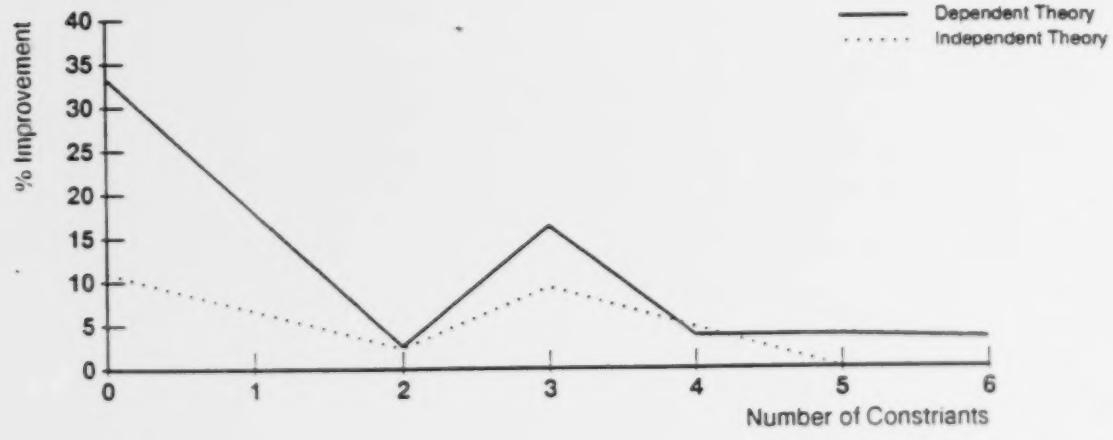


Figure 6.6: Example Base with 25 Events

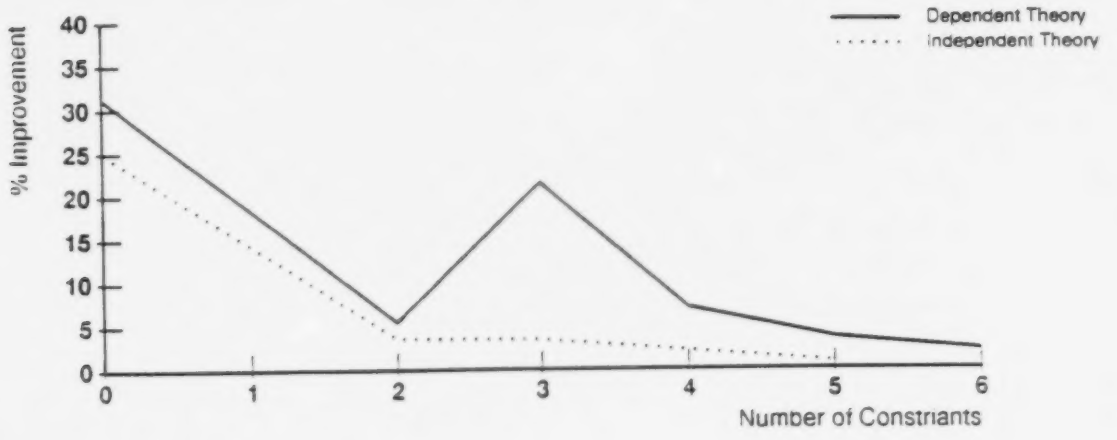


Figure 6.7: Example Base with 50 Events

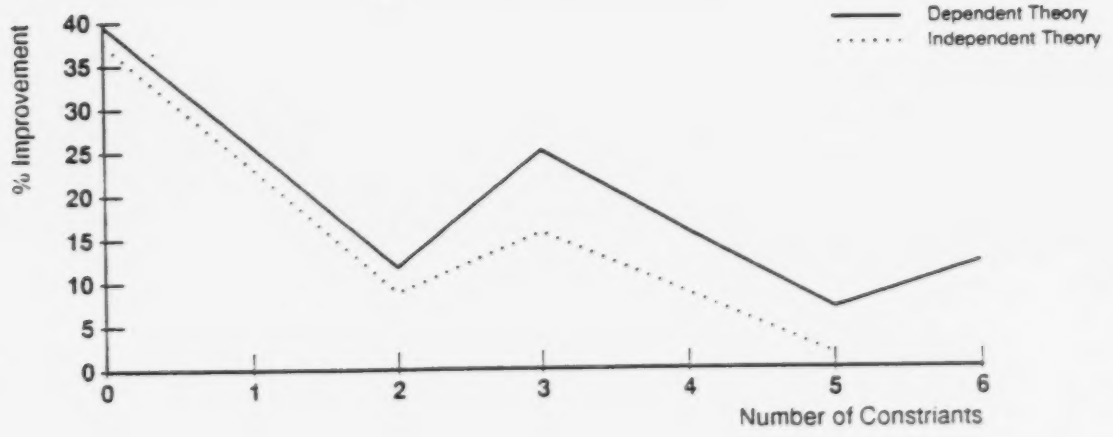


Figure 6.8: Example Base with 75 Events

For example, the improved accuracy of the new knowledge base when 75 events are in the example base ranges from 4% to 9% better than when 25 events are in the example base.

Intuition might suggest that having a large proof structure to guide the selection of examples is always beneficial. In fact, this is not the case. If the partial-proof tree consists of subgoals that have little or no relation to each other, their presence can decrease system performance. This occurs because the larger proof structure limits the examples used for generalization. In the worst case, the examples the system actually needs for generalization are eliminated from the collected set by unrelated subgoals. As more context is defined for the independent domain theory problems, the accuracy improvements by the induced rule diminish. For the dependent theory, the improvements are generally better.

The sharp fluctuations in the plots are a result of some rules covering more examples than other rules. The learning of rules that set default values for subgoals is hindered by the use of the context. The system learns rules that are too specific. As more examples are presented to the system, these rules can be generalized. The fluctuations are so dramatic in these small domains because there are few partial-proof trees that can be constructed for a given context size. When a default rule is the predominate rule induced for a context size, its performance dominates the plotted values.

The context size is the number of constraints placed on the collection of examples for learning. Generally it is the number of grounded leaf nodes in the partial-proof tree[1].

6.1.2 No Context Results

The KBL1 system requires rule knowledge and expends great effort in deriving a context. It is prudent to ask if the effort is worthwhile. In this experiment, the system used all the examples in the example base, not just those covered by a context. This was compared with the results of the dependent theory presented previously. It is important to realize that the partial-proof tree is still required to construct the binding portion of the new rule. Pure inductive systems have no mechanism for deriving those values.

[1]The independent domain produces a context of size five when rules 2 and 5 are used together. Both rules contain (= ?y (binding a)) and the system is able to unify the variable bound to ?y in each case, because each variable is set to a.

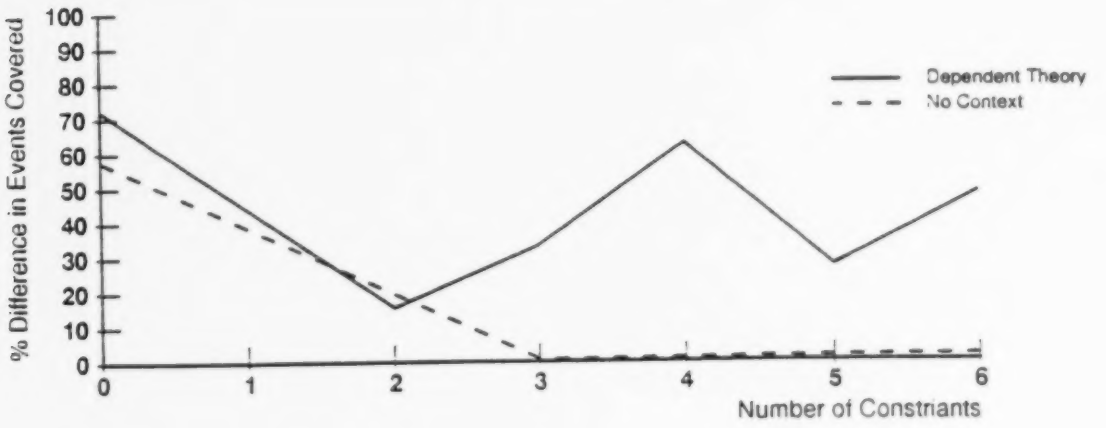


Figure 6.9: No Context Difference Results with 75 examples

The testing procedure was the same as described earlier. The complete dependent domain theory rules were used for the test. The best results for the no-context case occurred with 75 examples in the example base. This is consistent with results in SBL indicating such systems perform better with larger numbers of examples. These results are presented in Figures 6.9 and 6.10. For the no-context case, a value for number of contexts does not exist for the plot. The number of contexts was determined from previous runs using a context and observing which rule was missing.

As expected, the rules produced without a context did much worse than rules induced with examples meeting context constraints. This provides justification for the extra effort expended by KBL1 in determining the context. These results are for a small domain. As larger domains are used, the gap in performance between using a context and not using one should increase.

6.2 Phone Domain

In this section, some results are presented for KBL1 with a larger domain theory. The domain is motivated by a problem phone companies are trying to be solve. The problem is to identify when a call placed on a credit card has a high chance of not being placed by the

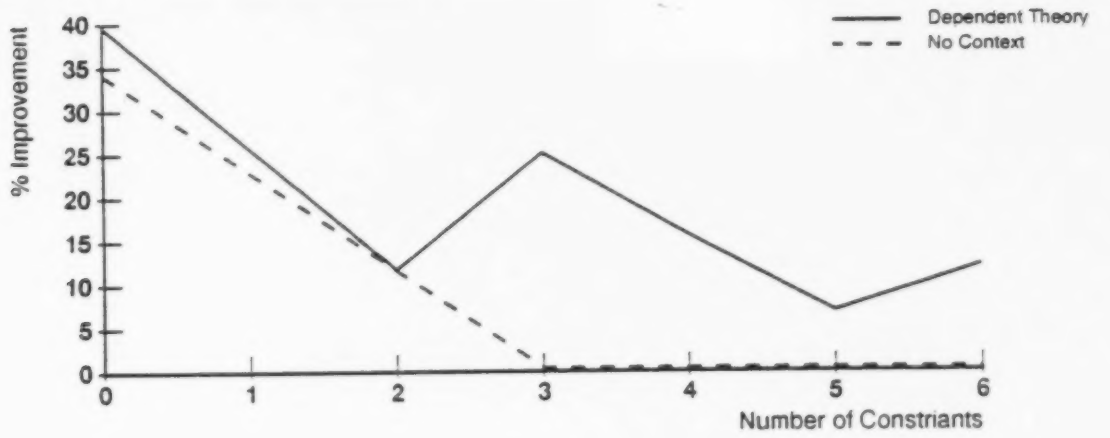


Figure 6.10: No Context Improvement Results with 75 Examples

card owner. That is, is this call placed by someone using a stolen card. A simple domain of thirty-three rules that contain the knowledge for a particular person. This person's home phone number is (391) 777-1111. They live in region 1 and don't make many phone calls to or from other regions. The area codes 801, 802, and 803 are known to be trouble areas. Many calls from that area are made on stolen cards. The top-level goal `bad-call` has six arguments. The first is the call currently being worked on. The remaining five arguments get instantiated with values indicating the chance that the call was from a stolen card for five different aspects. The items under consideration are: location call was made to, location call was made from, time of day for the call, frequency of calls to and from those numbers, and the length of the call. The domain knowledge follows:

1. `(bad-call ?x ?to-loc ?from-loc ?tod ?freq ?length) ←`
   `(susp-to-loc ?x ?to-loc) (susp-from-loc ?x ?from-loc) (susp-tod ?x ?tod)`
   `(susp-freq ?x ?freq) (susp-length ?x ?length)`

2. `(susp-to-loc ?x ?sp) ← (from-area-code ?x 391) (to-area-code ?x 391)`
   `(= ?sp (binding 0))`

3. (susp-to-loc ?x ?sp) ← (from-area-code ?x 391) (to-area-code ?x ?y)
(<> ?y 391) (to-region ?x 1) (= ?sp (binding 10))

4. (susp-to-loc ?x ?sp) ← (from-area-code ?x ?z)(<> ?z 391)(from-region ?x 1)
(to-region ?x 1) (= ?sp (binding 20))

5. (susp-to-loc ?x ?sp) ← (from-region ?x 1) (to-region ?x 2)
(to-area-code ?x ?z) (<> ?z 801) (= ?sp (binding 40))

6. (susp-to-loc ?x ?sp) ← (from-region ?x 1) (to-region ?x ?y) (<> ?y 1)
(to-area-code ?x 801) (= ?sp (binding 90))

7. (susp-to-loc ?x ?sp) ← (from-region ?x ?y) (<> ?y 1)(from-area-code ?x 801)
(to-region ?x 1) (= ?sp (binding 95))

8. (susp-to-loc ?x ?sp) ← (from-region ?x ?y) (<> ?y 1)(from-area-code ?x ?z)
(<> ?z 801) (to-region ?x 1) (= ?sp (binding 60))

9. (susp-to-loc ?x ?sp) ← (from-region ?x ?y) (<> ?y 1) (to-region ?x ?z)
(<> ?z 1) (= ?sp (binding 85))

10. (susp-from-loc ?x ?sp) ← (from-area-code ?x 391) (from-number ?x 7771111)
(= ?sp (binding 0))

11. (susp-from-loc ?x ?sp) ← (from-area-code ?x 391) (from-number ?x ?y)
(<> ?y 7771111)(= ?sp (binding 20))

12. (susp-from-loc ?x ?sp) ← (from-area-code ?x ?y) (member ?y (801 802 803))
(= ?sp (binding 90))

13. (susp-from-loc ?x ?sp) ← (from-region ?x 1) (from-area-code ?x ?y)
(<> ?y 391) (= ?sp (binding 40))

14. (susp-from-loc ?x ?sp) ← (from-region ?x ?y) (<> ?y 1)
(from-area-code ?x ?z) (<> ?z 801) (= ?sp (binding 70))

15. (susp-tod ?x ?sp) — (call-tod ?x ?y) (< 17 ?y) (from-area-code ?x 391)
(from-number ?x 7771111) (= ?sp (binding 0))

16. (susp-tod ?x ?sp) ← (call-tod ?x ?y) (≤ ?y 17)(from-area-code ?x 391)
(from-number ?x 7771111) (= ?sp (binding 5))

17. (susp-tod ?x ?sp) ← (call-tod ?x ?y) (≤ ?y 17)(from-area-code ?x 391)
(from-number ?x ?z) (<> ?z 7771111) (= ?sp (binding 7))

18. (susp-tod ?x ?sp) ← (call-tod ?x ?y) (< 17 ?y)(from-area-code ?x 391)
(from-number ?x ?z) (<> ?z 7771111) (= ?sp (binding 17))

19. (susp-tod ?x ?sp) ← (from-region ?x ?y) (<> ?y 1) (to-number ?x 7771111)
(= ?sp (binding 20))

20. (susp-tod ?x ?sp) ← (from-region ?x ?y) (<> ?y 1) (to-number ?x ?z)
(<> ?z 7771111) (= ?sp (binding 45))

21. (susp-tod ?x ?sp) ← (from-region ?x 1) (from-area-code ?x ?y) (<> ?y 391)
(= ?sp (binding 60))

22. (susp-freq ?x ?sp) ← (from-area-code ?x 391) (to-area-code ?x 391)
(= ?sp (binding 0))

23. (susp-freq ?x ?sp) ← (to-region ?x ?y) (<> ?y 1) (freq-to-number ?x ?z)
(< ?z 5) (= ?sp (binding 90))

24. (susp-freq ?x ?sp) ← (to-region ?x ?y) (<> ?y 1) (freq-to-number ?x ?z)
(≥ ?z 5) (= ?sp (binding 50))

25. (susp-freq ?x ?sp) ← (from-area-code ?x 391) (to-area-code ?x ?y)
(<> ?y 391)(freq-to-number ?x ?z) (< ?z 10) (to-region ?x 1)
(= ?sp (binding 10))

26. (susp-freq ?x ?sp) ← (from-area-code ?x 391) (to-area-code ?x ?y)
(<> ?y 391)(freq-to-number ?x ?z) (< 10 ?z) (to-region ?x 1)
(= ?sp (binding 5))

27. (susp-freq ?x ?sp) ← (from-area-code ?x ?y) (<> ?y 391)
(freq-to-number ?x ?z) (< 10 ?z) (from-region ?x 1)(= ?sp (binding 85))

28. (susp-freq ?x ?sp) ← (from-area-code ?x ?y) (<> ?y 391)
(freq-to-number ?x ?z) (≤ ?z 10) (from-region ?x 1) (= ?sp (binding 15))

29. (susp-freq ?x ?sp) ← (from-region ?x ?z) (<> ?z 1) (to-region ?x 1)
(= ?sp (binding 95))

30. (susp-length ?x ?sp) ← (from-area-code ?x 391) (= ?sp (binding 0))

31. (susp-length ?x ?sp) ← (from-region ?x ?y) (<> ?y 1) (duration ?x ?z)
(< ?z 5) (= ?sp (binding 10))

32. (susp-length ?x ?sp) ← (from-region ?x ?y) (<> ?y 1) (duration ?x ?z)
(≤ 5 ?z) (= ?sp (binding 90))

33. (susp-length ?x ?sp) ← (from-area-code ?x ?y) (<> ?y 391)
(from-region ?x 1) (= ?sp (binding 50))

Using this domain, the KBL1 system was given different sized example bases to verify that more examples do increase the system's performance. The examples were produced by a program generating the data. For each of the following experiments, a set of rules for a single consequent was removed from the knowledge base. Sets of rules were removed for **susp-from-loc**, **susp-tod**, **susp-freq**, and **susp-length**. For each incomplete knowledge base, the system was provided with example bases of 10, 25, 50, and 100 example/solution pairs. In each situation (a particular incomplete knowledge base and particular sized example base), the system was presented with 25 examples to be used for learning. Each situation was performed four different times. The results for each separate knowledge base are shown in Figures 6.11, 6.12, 6.13, and 6.14. As each example is presented, the system slowly adds knowledge to improve the accuracy of the performance element. As these four graphs indicate, the system performs better when provided with more initial examples.

The KBL1 system appears to be improving accuracy very slowing. The system is very conservative and produces rules using maximally specific generalizations. If the system took a less conservative approach to rule induction, the accuracy could improve much faster but the rules produced would not be as pleasing to engineers. Also, the system does not necessarily learn a new rule for every trial example presented. If the example is already covered by the knowledge base, no rules are created. In addition there are situations when the system cannot

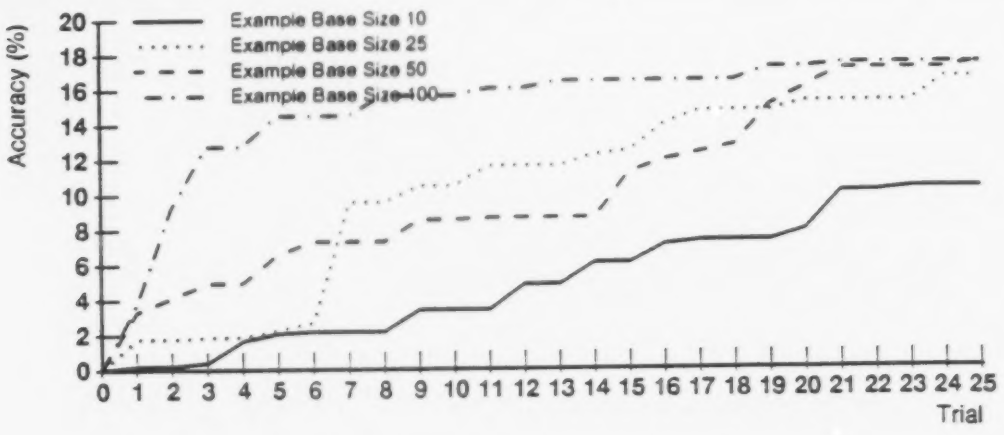

Example Base Size 10
Example Base Size 25
Example Base Size 50
Example Base Size 100
Accuracy (%)
0 2 4 6 8 10 12 14 16 18 20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Trial

Figure 6.11: Accuracy for KBL1 with `susp-from-loc` Removed from Phone Domain

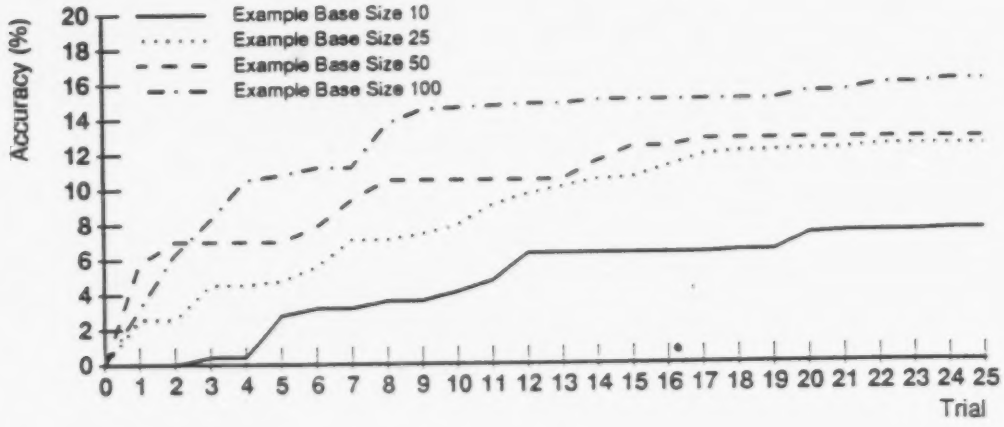

Example Base Size 10
Example Base Size 25
Example Base Size 50
Example Base Size 100
Accuracy (%)
0 2 4 6 8 10 12 14 16 18 20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Trial

Figure 6.12: Accuracy for KBL1 with `susp-tod` Removed from Phone Domain

learn a new rule even if the trial example is not covered. If the example base does not contain examples matching the context, no learning can occur.

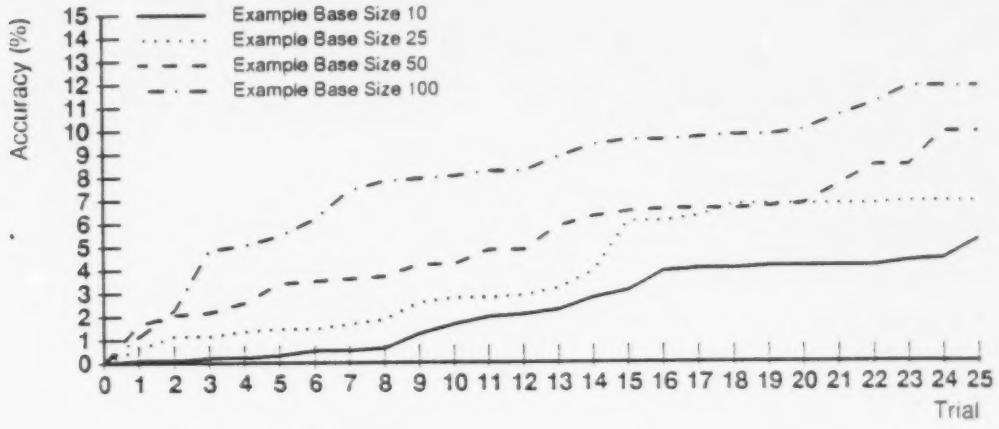


Figure 6.13: Accuracy for KBL1 with `susp-freq` Removed from Phone Domain

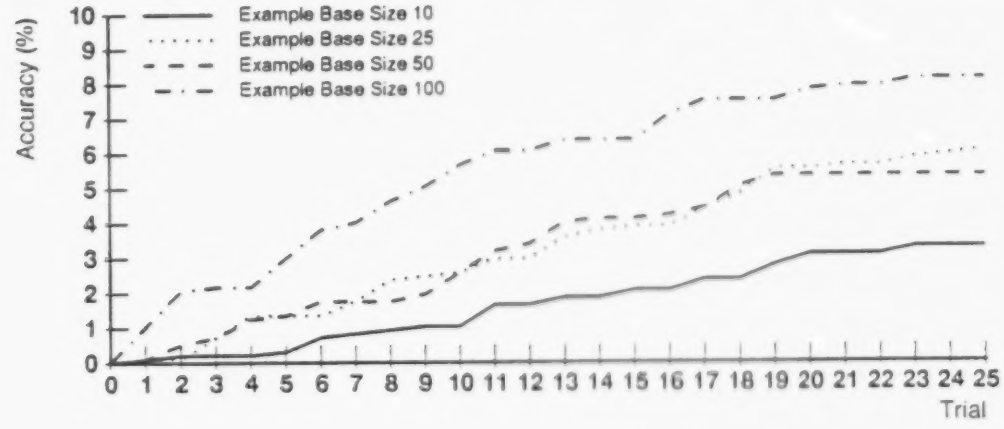


Figure 6.14: Accuracy for KBL1 with `susp-length` Removed from Phone Domain

Chapter 7

Conclusion

Knowledge-based learning is a new approach to integrating deductive and inductive learning. Knowledge-based learning uses the domain knowledge to guide the collection of examples for learning, not just to indicate that learning needs to occur. Knowledge-based learning also addresses issues that are important when constructing knowledge-based systems for engineering problems, such as finding mathematical formulas to describe events. The first section presents some open issues in knowledge-based learning. There are many areas where research can be expanded within this framework. The second section presents a summary of this work.

7.1 Future Work

There are a number of areas where the work on knowledge-based learning can be expanded. This is just the initial study of combining various learning techniques. One of the first areas requiring further investigation is determining how to define the appropriate context better than is currently done. It is clear that the context, the knowledge used to guide the induction, is very important. However, not all the knowledge in a proof tree is necessarily pertinent to the unsolved subgoal. It is likely that some of the branches are completely unrelated to the problem being solved. Other goals might be important, but the exact method used to prove them is insignificant. If the best context could be found, the most relevant examples could be collected for use in inducing the new rule.

Currently, the KBL approach is focused on learning domain knowledge. It seems natural to apply the same approach to control knowledge as well. Obviously, KBL could be applied to

learning control knowledge, if it is consider the domain. But a more interesting area to pursue is to have the system learn both control and domain knowledge at the same time. The examples collected for use inducing a domain knowledge rule need not be the same as the examples used to learn the control knowledge. At this point it is unclear whether learning at the control level should occur at the same time as learning at the domain level, or if they are separate problems that cannot benefit from knowledge used at the other level.

KBL improves the performance of a knowledge base system slowly due to the conservative nature of the rule induction. While engineers like rules that use many attributes, finding an induction module that produces rules more general than characteristic descriptions and more specific than discriminant descriptions would be beneficial. The maximally specific rules now produced by the system are good if an expert generalizes them before they are added to the knowledge base. A more powerful mechanism to focus the attributes included in the rules could perform the same task. Using general knowledge to guide the selection of useful attributes for a specific rule is a difficult task.

The KBL approach can be extended to be more integrated with deductive learning. The approach currently uses deduction to construct the partial-proof trees. The system could be extended to deductively generalize the proofs created with the induced rules.

Another area that deserves future attention is combining analogy with a system like KBL1. It seems clear that engineers use analogy when problem solving. Inductive leaps are also used in making the mappings from a base case to a target case in analogy, but the inductive leap is guided by previously held knowledge. Research into using knowledge to guide the mapping process could prove very rewarding.

Research in determining the applicability of the KBL methodology to specific domains and specific stages of knowledge base completion is important. As first mentioned in this thesis, a knowledge engineer can get most of the desired performance from a knowledge base very quickly (Figure 7.1). Empirical learning techniques are useful when the knowledge base is first being constructed. Analytical learning is useful when the knowledge base is nearly complete. Knowledge-based learning is useful when the knowledge base is partially complete. Future work will determine exactly when KBL should be applied. It is suspected that KBL is useful after the structure of the knowledge base is complete, but while many rules still need to be added. Knowing the exact time when KBL becomes most effective for specific domains would be useful.

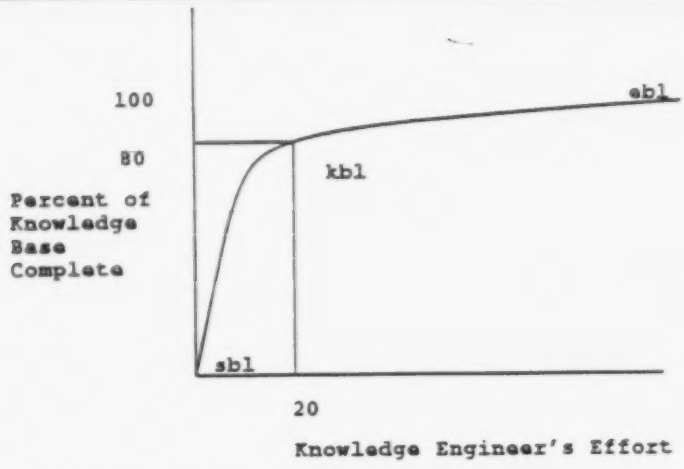


Figure 7.1: Knowledge Engineer's Effort Required to Achieve Desired Performance

It is also believed, that once the knowledge base is complete enough to warrant its use. KBL could be used indefinitely. Knowledge bases are never finished. The parameter values to the KBL system would need adjusted to reflect the confidence in the existing knowledge base. That is, the system might not change existing rules as easily as it adds new ones once the old rules had performed well in a number of situations.

Knowledge-based learning will work in different domains with varying degrees of success. Ideally, a knowledge engineer could determine the appropriateness of KBL for a domain before using it. To determine the applicability of KBL for a domain, the characteristics of the domain must be understood before the knowledge base is constructed, an interesting research topic in itself. KBL is useful when the domain consists of dependent rules. If results calculated for a problem use results calculated previously, KBL will be more effective. Future research should focus on determining the likely success of KBL verses other machine learning techniques for domains with distinct characteristics.

7.2 Final Summary

In this work, a methodology of combining inductive and deductive learning has been presented and explored. This methodology has been developed to work in engineering domains and with all the types of data that are used in those domains. It has been shown that existing

domain knowledge can be used to guide the induction of new domain knowledge from examples. Additionally, because of the tight coupling of knowledge with induction in the KBL approach, the produced rules can be incorporated into the existing knowledge base. The rules produced by pure inductive techniques do not conform to an existing knowledge base.

Knowledge-based learning is a general approach that requires the use of an incomplete domain model to guide the induction of new rules. The system induces new rules when it is attempting to solve a problem and the current knowledge base lacks the appropriate knowledge to compute a solution. Knowledge-based learning converts knowledge from examples into an explicit form that can be used by the performance element.

A knowledge-based learning system is composed of a number of components. The required components are:

- **Partial-Proof Tree Constructor.**

  This module must be able to select the best partial proofs for the problem. Obtaining the correct partial proof is important because it is used to retrieve the examples from an example base.

- **Example Collector.**

  This module takes the partial-proof tree and uses it to collect the examples most relevant to the current problem.

- **Missing Value Determiner.**

  A module is needed that can take an example/solution pair and determine what set of values is possible for a variable in the partial-proof structure. The system must be able to work backwards from the solution of the final goal to the point where a rule is missing. The system hypothesizes the wanted result from the missing rule. Conceptually, this is a constraint propagation system that can handle ranges, not just single values, and is able to create new possible worlds as needed.

- **Partitioner.**

  A partitioning module is required to divide the collected examples into useful groups. The groups used for KBL help determine the rule consequent binding in addition to the rule antecedents. This partitioner must be able to deal with ranges of values, not just single examples.

- **Consequent Inducer.** This module must be able to compute what binding should be assigned to the missing variable of the subgoal. It uses the results of the partitioner or works closely with it.

- **Antecedent Inducer.** This module produces the subgoals for the new rule. For engineering applications, most specific generalizations of the events in a partition seem to give the best results.

All of these functions must exist in a KBL system. However, nothing prevents one module from performing more than one of the tasks.

As research in machine learning continues, better tools will become available to use as modules to KBL systems. In order for machine learning systems to be of assistance in the construction of knowledge-based systems, they must be capable of learning both inductively and deductively. KBL is a first step in constructing such learning systems for engineering domains.

Appendix A

Engineering Test With ID3

In this appendix, some test results are presented that confirm the notion that simple. empirical learning systems are inadequate for learning in engineering domains. Engineering problems are complex with many subgoals of the problem interacting with other subgoals. Simple empirical learning systems are not (always) able to detect these interactions.

The ID3 algorithm [Quinlan, 1986] constructs a decision tree based upon the input examples provided. The classified examples are used to determine the relevant attributes for determining differences between classes. The constructed decision tree can be used to categorize new examples. An example's attribute value is tested at each node of the tree and the branch corresponding to the value is taken. Eventually a leaf node is reached that contains the classification for the example. ID3 produces highly accurate trees, the output of the system is often used as a bench mark by other systems.

For this experiment, generated examples for the PHAST system were used by ID3 to construct a decision tree. Each example contains 110 attributes. These attributes describe the traffic intersection for the problem and the characteristics of the lanes in the intersection. Figure A.1 presents the description from one example.

The top level goal for the PHAST system is a 31 place predicate, `DESIGN-SIGNAL`. The output from the PHAST system is an instantiated version of `DESIGN-SIGNAL` such as (`DESIGN-SIGNAL`
`YES 448 448/1199 NO 0 0 NO 0 0 YES 855`
`171/241 NO 0 0 NO 0 0 448 448/1199 855 171/241 7824925/6646057 75 2 24 0 0 45 0`
`0`). The `YES` and `NO` values of the result indicate if a particular type of phase (noted by that

((LANE-LEFT-TURN-VOLUME L6 43) (LANE-RIGHT-TURN-VOLUME L6 50) (LANE-THRU-VOLUME L6 351) (LANE-LEFT-TURN-VOLUME L5 42) (LANE-RIGHT-TURN-VOLUME L5 50) (LANE-THRU-VOLUME L5 358) (LANE-RIGHT-TURN-VOLUME L4 93) (LANE-THRU-VOLUME L4 304) (LANE-LEFT-TURN-VOLUME L3 59) (LANE-THRU-VOLUME L3 318) (LANE-RIGHT-TURN-VOLUME L2 95) (LANE-THRU-VOLUME L2 307) (LANE-LEFT-TURN-VOLUME L1 53) (LANE-THRU-VOLUME L1 315) (LANE-LEFT-TURN-VOLUME L4 0) (LANE-RIGHT-TURN-VOLUME L3 0) (LANE-LEFT-TURN-VOLUME L2 0) (LANE-RIGHT-TURN-VOLUME L1 0) (APPROACH AP-EAST) (APPROACH-NAME AP-EAST EASTBOUND) (APPROACH-TYPE AP-EAST EASTBOUND) (APPROACH-ORIENTATION AP-EAST EW) (APPROACH-NUMBER-OF-LANES AP-EAST 2) (APPROACH-OPPOSING-APPROACH AP-EAST WESTBOUND) (APPROACH-INTERSECT-APP-TO-RIGHT AP-EAST NORTHBOUND) (APPROACH-INTERSECT-APP-TO-LEFT AP-EAST SOUTHBOUND) (APPROACH AP-WEST) (APPROACH-NAME AP-WEST WESTBOUND) (APPROACH-TYPE AP-WEST WESTBOUND) (APPROACH-ORIENTATION AP-WEST EW) (APPROACH-NUMBER-OF-LANES AP-WEST 2) (APPROACH-OPPOSING-APPROACH AP-WEST EASTBOUND) (APPROACH-INTERSECT-APP-TO-RIGHT AP-WEST SOUTHBOUND) (APPROACH-INTERSECT-APP-TO-LEFT AP-WEST NORTHBOUND) (APPROACH AP-NORTH) (APPROACH-NAME AP-NORTH NORTHBOUND) (APPROACH-TYPE AP-NORTH NORTHBOUND) (APPROACH-ORIENTATION AP-NORTH NS) (APPROACH-NUMBER-OF-LANES AP-NORTH 1) (APPROACH-OPPOSING-APPROACH AP-NORTH SOUTHBOUND) (APPROACH-INTERSECT-APP-TO-RIGHT AP-NORTH WESTBOUND) (APPROACH-INTERSECT-APP-TO-LEFT AP-NORTH EASTBOUND) (APPROACH AP-SOUTH) (APPROACH-NAME AP-SOUTH SOUTHBOUND) (APPROACH-TYPE AP-SOUTH SOUTHBOUND) (APPROACH-ORIENTATION AP-SOUTH NS) (APPROACH-NUMBER-OF-LANES AP-SOUTH 1) (APPROACH-OPPOSING-APPROACH AP-SOUTH NORTHBOUND) (APPROACH-INTERSECT-APP-TO-RIGHT AP-SOUTH EASTBOUND) (APPROACH-INTERSECT-APP-TO-LEFT AP-SOUTH WESTBOUND) (LANE L1) (LANE-NAME L1 1) (LANE-TYPE L1 LEFT-THRU) (LANE-APPROACH L1 EASTBOUND) (LANE-LANE-WIDTH L1 11) (LANE-HEAVY-VEHICLES L1 5) (LANE-GRADE L1 0) (LANE-PARKING-MOVEMENTS L1 0) (LANE-BUS-STOPS L1 0) (LANE-AREA L1 0.9) (LANE L2) (LANE-NAME L2 2) (LANE-TYPE L2 THRU-RIGHT) (LANE-APPROACH L2 EASTBOUND) (LANE-LANE-WIDTH L2 11) (LANE-HEAVY-VEHICLES L2 5) (LANE-GRADE L2 0) (LANE-PARKING-MOVEMENTS L2 0) (LANE-BUS-STOPS L2 0) (LANE-AREA L2 0.9) (LANE L3) (LANE-NAME L3 3) (LANE-TYPE L3 LEFT-THRU) (LANE-APPROACH L3 WESTBOUND) (LANE-LANE-WIDTH L3 11) (LANE-HEAVY-VEHICLES L3 5) (LANE-GRADE L3 0) (LANE-PARKING-MOVEMENTS L3 0) (LANE-BUS-STOPS L3 0) (LANE-AREA L3 0.9) (LANE L4) (LANE-NAME L4 4) (LANE-TYPE L4 THRU-RIGHT) (LANE-APPROACH L4 WESTBOUND) (LANE-LANE-WIDTH L4 11) (LANE-HEAVY-VEHICLES L4 5) (LANE-GRADE L4 0) (LANE-PARKING-MOVEMENTS L4 0) (LANE-BUS-STOPS L4 0) (LANE-AREA L4 0.9) (LANE L5) (LANE-NAME L5 5) (LANE-TYPE L5 LEFT-THRU-RIGHT) (LANE-APPROACH L5 NORTHBOUND) (LANE-LANE-WIDTH L5 15) (LANE-HEAVY-VEHICLES L5 8) (LANE-GRADE L5 0) (LANE-PARKING-MOVEMENTS L5 0) (LANE-BUS-STOPS L5 0) (LANE-AREA L5 0.9) (LANE L6) (LANE-NAME L6 6) (LANE-TYPE L6 LEFT-THRU-RIGHT) (LANE-APPROACH L6 SOUTHBOUND) (LANE-LANE-WIDTH L6 15) (LANE-HEAVY-VEHICLES L6 8) (LANE-GRADE L6 0) (LANE-PARKING-MOVEMENTS L6 0) (LANE-BUS-STOPS L6 0) (LANE-AREA L6 0.9))

Figure A.1: Typical PHAST Example

position) needs to be used for this intersection. If that phase is required, then the volume and time for the phase are computed. The remaining numbers indicate overall statistics for the intersection, such as critical volumes and phase lengths.

In order for a decision tree to be useful for such data, each result must be considered its own class. That is, the phase times for an example are the values of interest and all the values must be correct. 192 examples were used by ID3 in constructing a decision tree. The tree produced contained 191 leaves. The decision tree was then used to classify 64 previously unseen examples. None of the 64 examples was classify correctly; the system was 0% accurate on unseen examples. This result does not mean that ID3 is a bad system. Rather, it indicates that ID3 cannot handle the complexities of the engineering domain. There are complex, formula relations between the input facts and out phase times of the PHAST system. The version of ID3 used for the test handles real valued attributes, but cannot find mathematical relationships between the attributes. This weakness does not allow ID3 to perform well on this task.

To provide an understanding for the complexity of the decision tree produced by ID3, it is included next. For each node of the tree, the decision is provided. If there is a leaf immediately below the branch, the value and number of examples (in parentheses) are provided.

```
LANE-THRU-VOLUME-L4  < 765/2:
|  LANE-LEFT-TURN-VOLUME-L6  < 77/2:
|  |  LANE-RIGHT-TURN-VOLUME-L5  < 55:
|  |  |  LANE-THRU-VOLUME-L6  < 355:
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 111/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 103/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 37:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 101/2: 30 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 101/2: 10 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 37: 33 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 103/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 107/2: 28 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 107/2: 139 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2: 134 (1)
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 111/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 115/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 37:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 85/2: 17 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 85/2: 8 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 37: 104 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 115/2:
```

```
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2: 26 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 29 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 11 (1)
|  |  |  LANE-THRU-VOLUME-L6 >= 355:
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 55:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 358: 18 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 358: 20 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 98 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2:
|  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 52: 113 (1)
|  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 52: 48 (2)
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 55:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2:
|  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 113/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 111/2: 12 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 111/2: 103 (1)
|  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 113/2: 142 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 57: 32 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 57: 102 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 123 (1)
|  |  LANE-RIGHT-TURN-VOLUME-L5 >= 55:
|  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 55:
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 53:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L4  < 787/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2: 112 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 51: 42 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 51: 41 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L4 >= 787/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 143 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2: 135 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2: 121 (1)
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 53:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 107/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2: 15 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 23 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 7 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 107/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 36: 140 (1)
```

```
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 36:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 707/2: 125 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 707/2: 24 (1)
|  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 55:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 117/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 57: 106 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 57: 120 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2: 100 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 117/2:
|  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 703/2: 114 (1)
|  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 703/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 353: 116 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 353: 9 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 57:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 83/2: 35 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 83/2: 21 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2: 27 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 57:
|  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 117/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 115/2: 4 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 115/2: 13 (1)
|  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 117/2: 119 (1)
|  LANE-LEFT-TURN-VOLUME-L6 >= 77/2:
|  |  LANE-LEFT-TURN-VOLUME-L5  < 40:
|  |  |  LANE-RIGHT-TURN-VOLUME-L4  < 783/2:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 105/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 79/2: 46 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 79/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 103/2: 14 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 103/2: 40 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 105/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 79/2: 5 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 79/2: 38 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2: 16 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 73/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 43: 37 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 43:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 53: 3 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 53: 31 (1)
```

```
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 73/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 85/2: 34 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 85/2: 47 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 137 (1)
|  |  |  LANE-RIGHT-TURN-VOLUME-L4 >= 783/2:
|  |  |  |  LANE-THRU-VOLUME-L5  < 713/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 53:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 42: 130 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 42:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 51: 105 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 51: 109 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 53:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 109/2: 133 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 109/2: 97 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2: 136 (1)
|  |  |  |  LANE-THRU-VOLUME-L5 >= 713/2:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 713/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 113/2: 101 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 113/2: 110 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 124 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 713/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 41: 127 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 41:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 43: 138 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 43: 99 (1)
|  |  LANE-LEFT-TURN-VOLUME-L5 >= 40:
|  |  |  LANE-LEFT-TURN-VOLUME-L5  < 42:
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 55:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L4  < 247:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 42: 44 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 42:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2: 36 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 6 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L4 >= 247:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 52: 107 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 52: 132 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2: 129 (1)
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 55:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 41:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 79/2: 25 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 79/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 117/2: 115 (1)
```

```
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 117/2: 2 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 41:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 42: 118 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 42:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2: 108 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 22 (1)
|  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 42:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L1  < 69:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 55:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2: 122 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 85/2: 131 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 85/2: 1 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 55:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2: 43 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 355: 19 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 355: 128 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L1 >= 69:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 79/2: 45 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 79/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 107/2: 126 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 107/2: 39 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2: 117 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 109/2: 111 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 109/2: 141 (1)
LANE-THRU-VOLUME-L4 >= 765/2:
|  LANE-RIGHT-TURN-VOLUME-L6  < 109/2:
|  |  LANE-RIGHT-TURN-VOLUME-L5  < 109/2:
|  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 53:
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 103/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 38:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2: 77 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 715/2: 49 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 715/2: 78 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 38:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 79/2: 64 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 79/2: 72 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2: 148 (1)
|  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 103/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 42:
```

```
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 38: 66 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 38: 168 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2: 187 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 42:
|  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 355: 159 (1)
|  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 355:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 715/2: 84 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 715/2: 170 (1)
|  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 53:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 354:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 39:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 51 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 53 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 39: 154 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 354:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 36: 96 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 36:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2: 191 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2: 52 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 43:
|  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 709/2: 179 (1)
|  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 709/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 358: 150 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 358: 65 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 43:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2: 145 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 352: 91 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 352: 173 (1)
|  |  LANE-RIGHT-TURN-VOLUME-L5 >= 109/2:
|  |  |  LANE-RIGHT-TURN-VOLUME-L4  < 489/2:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 39:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 354:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2: 59 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 105/2: 63 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 105/2: 81 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 354:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 93 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 53: 92 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 53: 68 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 39:
```

```
|  |  |  |  |  LANE-THRU-VOLUME-L5  < 711/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 42:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 52: 95 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 52: 50 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 42: 89 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L5 >= 711/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 103/2: 73 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 103/2: 88 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 61 (1)
|  |  |  LANE-RIGHT-TURN-VOLUME-L4 >= 489/2:
|  |  |  |  LANE-THRU-VOLUME-L6  < 711/2:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 353:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 42:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 51: 156 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 51: 160 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 42: 146 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 353:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 182 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 39: 180 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 39: 161 (1)
|  |  |  |  LANE-THRU-VOLUME-L6 >= 711/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 40:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 77/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 149 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 189 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 77/2: 164 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 40:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 85/2: 166 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 85/2:
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 715/2: 183 (1)
|  |  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 715/2: 147 (1)
|  LANE-RIGHT-TURN-VOLUME-L6 >= 109/2:
|  |  LANE-LEFT-TURN-VOLUME-L6  < 79/2:
|  |  |  LANE-RIGHT-TURN-VOLUME-L5  < 53:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 79/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 111/2: 178 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 111/2: 177 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2: 82 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 38: 80 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 38:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 113/2: 94 (1)
```

```
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 113/2: 67 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 79/2:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 705/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 56: 151 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 56: 75 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 165 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 705/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2: 190 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 87/2: 153 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 87/2: 144 (1)
|  |  |  LANE-RIGHT-TURN-VOLUME-L5 >= 53:
|  |  |  |  LANE-THRU-VOLUME-L6  < 703/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2:
|  |  |  |  |  |  LANE-THRU-VOLUME-L6  < 701/2: 184 (1)
|  |  |  |  |  |  LANE-THRU-VOLUME-L6 >= 701/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 75/2: 169 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 75/2: 90 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 75/2: 167 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 75/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 111/2: 55 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 111/2: 79 (1)
|  |  |  |  LANE-THRU-VOLUME-L6 >= 703/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L4  < 195/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 36: 70 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 36:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2: 56 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 69 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L4 >= 195/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 73/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 71/2: 57 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 71/2: 158 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 73/2: 155 (1)
|  |  LANE-LEFT-TURN-VOLUME-L6 >= 79/2:
|  |  |  LANE-THRU-VOLUME-L6  < 356:
|  |  |  |  LANE-THRU-VOLUME-L6  < 707/2:
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 43:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 41: 163 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 41:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 57: 71 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 57: 85 (1)
|  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 43:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 115/2: 185 (1)
```

```
|  |  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 115/2: 172 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 83 (1)
|  |  |  |  LANE-THRU-VOLUME-L6 >= 707/2:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 709/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 81/2: 87 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 81/2: 186 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2: 171 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 709/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 87/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2: 58 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2: 175 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 87/2: 181 (1)
|  |  |  LANE-THRU-VOLUME-L6 >= 356:
|  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 81/2:
|  |  |  |  |  LANE-THRU-VOLUME-L6  < 358:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 41:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5  < 73/2: 54 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 73/2: 162 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 41: 188 (1)
|  |  |  |  |  LANE-THRU-VOLUME-L6 >= 358:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 56: 176 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 56: 76 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2: 74 (1)
|  |  |  |  LANE-LEFT-TURN-VOLUME-L5 >= 81/2:
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 115/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 83/2: 86 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 83/2:
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 43: 62 (1)
|  |  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 43: 60 (1)
|  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 115/2:
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6  < 85/2:
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6  < 117/2: 174 (1)
|  |  |  |  |  |  |  LANE-RIGHT-TURN-VOLUME-L6 >= 117/2: 152 (1)
|  |  |  |  |  |  LANE-LEFT-TURN-VOLUME-L6 >= 85/2: 157 (1)
```

Bibliography

[Ahn *et al.*, 1987] W. Ahn, R. J. Mooney, W. F. Brewer, and G. F. DeJong. Schema acquisition from one example: Psychological evidence for explanation-based learning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pages 50–57, July 1987.

[Angluin and Laird, 1988] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[Barr and Feigenbaum, 1981] A. Barr and E. Feigenbaum. *The Handbook of Artificial Intelligence*, volume 1. HeurisTech Press, Stanford, CA, 1981.

[Buchanan and Feigenbaum, 1978] B. G. Buchanan and E. A. Feigenbaum. Dendral and meta-dendral: Their applications dimension. *Artificial Intelligence*, 11:5–24, August 1978. Special issue on Applications to the Sciences and Medicine. Edited by N.S. Sridharan.

[Buchanan *et al.*, 1976] B. G. Buchanan, D.H. Smith, W.C. White, R. Gritter, E.A. Feigenbaum, J. Lederberg, and C. Djerassi. Applications of artificial intelligence for chemical inference, xxii, automatic rule-formation in mass spectrometry by means of the meta-dendral program. *J. Amer. Chem. Soc.*, 98:6168–6178, 1976.

[Buchanan *et al.*, 1983] B. G. Buchanan, D. Barstow, R. Bechtal, J. Bennett, W. Clancey, C. Kulikowski, T. Mitchell, and D. A. Waterman. Constructing an expert system. In F. Hayes-Roth, D. Waterman, and D. Lenat, editors, *Building Expert Systems*, chapter 5, pages 127–167. Addison Wesley Publishing Company, Inc., 1983.

[Chien, 1987] S. A. Chien. Simplifications in temporal persistence: An approach to the intractable domain theory problem in explanation-based learning. Master's thesis, Computer Science Department, University of Illinois, 1987.

[Chien, 1989] S.A. Chien. Learning by analyzing fortuitous occurances. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 249–251. Morgan Kaufmann Publishers, June 1989.

[Danyluk, 1987] A. P. Danyluk. The use of explanations for similarity-based learning. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 274–276. Milan, Italy, August 1987.

[Davis, 1982] R. Davis. Applications of meta-level knowledge to construction, maintenance and use of large knowledge bases. In R. Davis and D. Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 229–490. McGraw-Hill, 1982.

[DeJong and Mooney, 1986] Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.

[DeJong, 1990] Gerald DeJong. Explanation-based learning with plausible inferencing. Technical Report UIUCDCS-R-90-1577, Computer Science Department, University of Illinois, Urbana, IL, 1990.

[Dietterich and Michalski, 1983] T.G. Dietterich and R.S. Michalski. A comparative review of selected methods for learning from examples. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach*, volume 1, pages 41–82. Morgan Kaufmann Publishers, 1983.

[Dietterich, 1986] Thomas G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1(3):287–315, 1986.

[Falkenhainer and Michalski, 1986] Brian Falkenhainer and Ryszard S. Michalski. Integrating qualitative and quantitative discovery: The ABACUS system. *Machine Learning*, 1(4):367–401, 1986.

[Fawcett, 1989] T.E. Fawcett. Learning from plausible explanations. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 37–39. Morgan Kaufmann Publishers, June 1989.

[Feigenbaum, 1984] A. E. Feigenbaum. Leture at the First U.S.-China Joint Seminar on Automatio and Intelligent Systems, 1984. May 28 – June 1.

[Fisher, 1987] D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[Flann and Dietterich, 1986] N. S. Flann and T. G. Dietterich. Selecting appropriate representations for learning from examples. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 460–466, Philadelphia, PA, August 1986.

[Ham, 1982] Inyong Ham. Group technology. In Gavriel Salvendy, editor, *Handbook of Industrial Engineering*, pages 7.8.1–19. John Wiley & Sons, 1982.

[Hoff *et al.*, 1983] W. A. Hoff, Ryszard S. Michalski, and Robert E. Stepp. INDUCE 3: A program for learning structural descriptions from examples. Technical Report UIUCDCS-F-83-904, Computer Science Department, University of Illinois, Urbana, IL, 1983.

[Holder, 1988] L. B. Holder. Discovering substructure in examples. Master's thesis, Computer Science Department, University of Illinois, 1988.

[Holder, 1990] L. B. Holder. The general utility problem in machine learning. In *Proceedings of the Seventh International Conference on Machine Learning*, 1990.

[Holland, 1975] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[Keller, 1988] Richard M. Keller. Defining operationality for explanation-based learning. *Artificial Intelligence*, 35(2):227–241, June 1988.

[Kodratoff and Tecuci, 1987] Y. Kodratoff and G. Tecuci. What is an explanation in disciple? In *Proceedings of the Fourth International Machine Learning Workshop*, pages 160–166. Morgan Kaufmann Publishers, June 1987.

[Kolodner *et al.*, 1985] J. L. Kolodner, R. L. Simpson, and K. Sycara. A process model of case-based reasoning in problem solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 284–290. Morgan Kaufmann Publishers, Inc., 1985.

[Langley *et al.*, 1987] Pat Langley, Herbert A. Simon, Gary L. Bradshaw, and Jan M. Zytkow. *Scientific Discovery: Computational Explorations of the Creative Process*. MIT Press, Cambridge, 1987.

[Larson, 1977] J. Larson. *Inductive Inference in the Variable-Valued Predicate Logic System VL21: Methodology and Computer Implementation*. PhD thesis, Computer Science Department, University of Illinois, 1977.

[Lebowitz, 1986a] M. Lebowitz. Concept learning in a rich input domain: Generalization-based memory. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach*, volume 2, pages 193–214. Morgan Kaufmann Publishers, 1986.

[Lebowitz, 1986b] M. Lebowitz. Integrated learning: Controlling explanation. *Cognitive Science*, 10(2):219–240, 1986.

[Linkenheld *et al.*, 1989] J. Linkenheld, J. H. Garrett, Jr., and R. Benekohal. A knowledge-based system for the design of signalized intersections. In *First Conference on Expert Systems in Environmental and Urban Planning*, Boston, MA, September 1989.

[Lu and Chen, 1987] S. C-Y. Lu and K. Chen. A machine learning approach to the automatic synthesis of mechanistic knowledge for engineering decision making. *Journal of Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 1(2):109–118, 1987.

[Lu, 1989] S. C-Y. Lu. Machine learning techniques for group technology applications. *CIRP Annual*, 39(2), 1989.

[Matheus, 1989a] Christopher J. Matheus. *Feature Construction: An Analytical Framework and an Application to Decision Trees*. PhD thesis, University of Illinois at Urbana-Champaign, December 1989.

[Matheus, 1989b] C.J. Matheus. A constructive induction framework. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 474–475. Morgan Kaufmann Publishers, June 1989.

[Michalski and Chilausky, 1980] R. S. Michalski and R. L. Chilausky. Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing and expert system for soybean disease diagnosis. *Policy Analysis and Information Systems*. 4(2), 1980.

[Michalski and Stepp, 1983] R.S. Michalski and R.E. Stepp. Learning from observation: Conceptual clustering. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors. *Machine Learning: An AI Approach*, volume 1, pages 331–364. Morgan Kaufmann Publishers, 1983.

[Michalski, 1985] R. S. Michalski. Synthesis of optimal and quasi-optimal variable valued logic formulas. In *Proceedings of the 1975 International Symposium on Multiple-Valued Logic*, pages 76–87, Bloomington, IN, September 1985.

[Minton *et al.*, 1987] S. Minton, J.G. Carbonell, and O. Etzioni. Acquiring effective search control rules: Explanation-based learning in the prodigy system. In *Proceedings of the Fourth International Machine Learning Workshop*, pages 122–133. Morgan Kaufmann Publishers, June 1987.

[Mitchell *et al.*, 1983] T.M. Mitchell, P.E. Utgoff, and R. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach*, volume 1, pages 163–190. Morgan Kaufmann Publishers, 1983.

[Mitchell *et al.*, 1986] Thomas M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80, January 1986.

[Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[Mogensen, 1987] B. N. Mogensen. Goal-oriented conceptual clustering: The classifying attribute approach. Master's thesis, Computer Science Department, University of Illinois, 1987.

[Mooney and Ourston, 1989] R. Mooney and D. Ourston. Induction over the unexplained: Integrated learning of concepts with both explainable and conventional aspects. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 5–7. Morgan Kaufmann Publishers, June 1989.

[Mooney, 1988] R. J. Mooney. *A General Explanation-Based Learning Mechanism and its Application to Narrative Understanding*. PhD thesis, Computer Science Department, University of Illinois, 1988.

[O'Rorke, 1987] P. V. O'Rorke. *Explanation-Based Learning Via Constraint Posting and Propagation*. PhD thesis, Computer Science Department, University of Illinois, 1987.

[Pazzani *et al.*, 1986] M. J. Pazzani, M. Dyer, and M. Flowers. The role of prior causal theories in generalization. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 545–550, Philadelphia, PA, August 1986.

[Pazzani *et al.*, 1987] M. J. Pazzani, M. Dyer, and M. Flowers. Using prior learning to facilitate the learning of new causal theories. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 277–279, Milan, Italy, August 1987.

[Pazzani, 1985] M. J. Pazzani. Explanation and generalization based memory. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 323–328. Irvine, CA, August 1985.

[Pazzani, 1987] M.J. Pazzani. Inducing casual and social theories: A prerequisite for explanation-based learning. In *Proceedings of the Fourth International Machine Learning Workshop*, pages 230–241. Morgan Kaufmann Publishers, June 1987.

[Pitt and Reinke, 1988] L. Pitt and R. Reinke. Criteria for polynomial-time (conceptual) clustering. *Machine Learning*, 2(4):371–396, 1988.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[Rajamoney, 1988] S. A. Rajamoney. *Explanation-Based Theory Revision: An Appraoch to the Problem of Incomplete and Incorrect Theories.* PhD thesis, Computer Science Department, University of Illinois, 1988.

[Rendell, 1989] L. Rendell. Comparing systems and analyzing functions to improve constructive induction. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 461–464. Morgan Kaufmann Publishers, June 1989.

[Schlimmer and Granger, 1986] J.C. Schlimmer and R.H. Granger, Jr. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[Segre, 1987] A. M. Segre. *Explanation-based learning of generalized robot assembly plans.* PhD thesis, Computer Science Department, University of Illinois, 1987.

[Shavlik, 1988] J. W. Shavlik. *Generalizing the Structure of Explanations in Explanation-Based Learning.* PhD thesis, Computer Science Department, University of Illinois, 1988.

[Stepp and Michalski, 1986a] R.E. Stepp and R.S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach*, volume 2, pages 471–498. Morgan Kaufmann Publishers, 1986.

[Stepp and Michalski, 1986b] R.E. Stepp and R.S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An AI Approach*, volume 2, pages 471–498. Morgan Kaufmann Publishers, 1986.

[Stepp, 1984] R.E. Stepp. *Conjunctive Conceptual Clustering: A Methodology and Experimentation.* PhD thesis, Computer Science Department, University of Illinois, 1984.

[Tecuci and Kodratoff, 1989] G. Tecuci and Y. Kodratoff. Multi-strategy learning in nonhomogeneous domain theories. In *Proceedings of the Sixth International Machine Learning Workshop*, pages 14–16. Morgan Kaufmann Publishers, June 1989.

[Vere, 1977] Steven A. Vere. Induction of relational productions in the presence of background information. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 349–355. Proceedings of the International Joint Conference on Artificial Intelligence, 1977.

[Vere, 1980] Steven A. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14:139–164, 1980.

[Whitehall *et al.*, 1990] B. L. Whitehall, R. E. Stepp, and S. C y. Lu. Equation generation for clustering. In *Proceedings of The Third International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, pages 999–1004, Charleston, SC, July 1990.

[Whitehall *et al.*, 1991] B. L. Whitehall, S. C-Y. Lu, and R. E. Stepp. Caq: A machine learning tool for engineering. *The International Journal for Artificial Intelligence in Engineering*, 1991.

[Whitehall, 1987] B. L. Whitehall. Substructure discovery in executed action sequences. Master's thesis, Computer Science Department, University of Illinois, 1987.

[Wilkins, 1987] David C. Wilkins. *Apprenticeship Learning Techniques for Knowledge Based Systems*. PhD thesis, Stanford University, 1987.

[Winston, 1984] Patrick H. Winston. *Artificial Intelligence*. Addison Wesley Publishing Company, Inc., Reading, MA, second edition, 1984.

Vita

Bradley Lane Whitehall was born on June 3rd, 1961 in Aledo, Illinois. He attended Northern Illinois University in DeKalb, Illinois where he received a B.S. Degree in Computer Science and Mathematics in 1983. Following graduation, he was employed by Software Architects, Inc., Chicago, Illinois as a database consultant. In the fall of 1985 he began his graduate studies in Computer Science at the University of Illinois at Urbana-Champaign. In January of 1986 he became a research assistant for Professor Robert Stepp in the Coordinated Science Lab. He completed his Masters of Science degree in October of 1987 with a thesis on discovering substructures in observed action sequences. He received the Ph.D. in Computer Science in October of 1990. He is currently a member of the KBERSL Lab at the University of Illinois in a post-doc position.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-90-1637	2.	3. Recipient's Accession No.
4. Title and Subtitle KNOWLEDGE-BASED LEARNING: INTEGRATION OF DEDUCTIVE AND INDUCTIVE LEARNING FOR KNOWLEDGE BASE COMPLETION			5. Report Date October 1990 6.
7. Author(s) Bradley Lane Whitehall			8. Performing Organization Rept. No. R-90-1637
9. Performing Organization Name and Address Dept. of Computer Science University of Illinois 1304 W. Springfield Ave. Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered Technical 14.

15. Supplementary Notes

16. Abstracts

To learn effectively, a system needs to use all the knowledge that is available. Explanation-based learning and similarity-based learning operate over a domain theory and a set of examples, respectively, but neither approach makes extensive use of both forms of knowledge. Many problems in engineering and other areas can provide a learning system with an incomplete domain theory and a limited set of examples. Knowledge-based learning uses knowledge in both forms to learn knowledge missing from the domain theory.

The knowledge-based learning approach is illustrated with two systems, KBLØ and KBL1. These system have been designed and implemented to work with domains requiring a representation of real numbers and mathematical formulas, such as engineering.

This research has shown, not only that it is possible to use a domain theory to guide induction using examples, but that when there are few examples available compared to the size of the problem space, the resulting rules are more accurate and stable than those from pure empirical techniques. In addition, knowledge-based learning algorithms free the user from selecting relevant examples and attributes for learning by using an incomplete domain theory to determine where knowledge needs to be added. A problem unsolved by the current domain knowledge helps to determine where new knowledge needs to be incorporated into the domain theory and what the context is for the learning. The context is used to select relevant examples from an example base and to reduce the number of attributes used during induction. With the control structure provided by knowledge-based systems, inductive learning can be used to extend an existing knowledge base.

17. Keywords

Machine learning
Integrated learning
Knowledge acquisition
Knowledge-Based Learning
Knowledge-base completion

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
unlimited	UNCLASSIFIED	162
163	20. Security Class (This Page) UNCLASSIFIED	22. Price

USCOMM-DC 40329-P71